Locked into stupidity: A smart lock security analysis

Kieron Turk
Gonville & Caius College

A dissertation submitted to the University of Cambridge in partial fulfilment of the requirements for the degree of Computer Science Tripos, Part III

University of Cambridge
Computer Laboratory
William Gates Building
15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom

Email: kst36@cam.ac.uk

May 4, 2023

Declaration

I, Kieron Turk of Gonville & Caius College, being a candidate for Computer Science Tripos, Part III, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose.

Total word count: 11753

Signed: 

Date: May 4, 2023

Acknowledgements

I would like to thank Ilia Shumailov for helping me to come up with the project, and Rosie Baish for suggesting the title. I would also like to thank my supervisors Dimitrije Erdeljan and Ross Anderson for their support throughout this project.

Locked into stupidity: A smart lock security analysis

Abstract

There are a large number of "smart" devices in users' homes. Developers are turning many standard household appliances into Internet of Things devices, often focusing on innovation. This often leads to a widespread absence of security in smart devices. This is especially dangerous for systems that need a focus on security, such as smart locks.

In this dissertation, I perform a security analysis of a pair of smart locks to attempt to find vulnerabilities in the systems that could be exploited by an attacker and identify common security problems that likely apply to other smart lock systems. The first lock connects to a mobile app over Wi-Fi, while the second uses Bluetooth Low-Energy; both also use passcodes, fingerprints, RFID cards, and a mechanical backup lock.

I find vulnerabilities in many aspects of both of the smart locks analysed. The majority of the vulnerabilities found are in the Bluetooth protocol; however, I have also exploited the Wi-Fi lock's setup process, both RFID unlock systems, and the physical security of each lock. These vulnerabilities highlight several broader issues in smart lock design, including the use of outdated and insecure components, difficulty implementing setup and cryptography in a secure manner, and failing to properly address both physical and cybersecurity simultaneously.

Word count: 11753

Contents

List of Figures

Chapter 1

Introduction

The Internet of Things has been expanding rapidly in recent years, with more and more household items becoming "smart" as time progresses. Many smart device creators are focused on innovation, aiming to be the first to connect a new device to the internet; however, very few IoT developers focus on security. Many do not consider security at all, leading to an expanse of vulnerabilities in internet-connected devices. Hackers can use mass-scanning tools such as ZMap[1] or Masscan[2] or services such as Shodan[3] to identify exposed devices over the internet. An attacker can then exploit these devices for a variety of malicious purposes.

The creation of smart locks bridges the worlds of cybersecurity and physical security. Smart locks require both their digital and physical aspects to be secure, however the creators of these devices often have a similar amount of interest in security to their fellow IoT developers. Companies that are transitioning to smart locks will often have experience in either physical security (from designing mechanical locks) or cybersecurity (from application development) but rarely understand both. This leads to many smart locks having issues in at least one aspect, which is a critical problem for smart lock owners. A vulnerability in any part of the device could potentially allow an attacker to gain access to the lock and everything it guards, or prevent a legitimate user from being able to operate the lock.

Manufacturers have few incentives to focus on the security of their products. Marketing most smart devices as secure does not make them sell more, however advertising all of their functionality will. For security products such as smart locks, vendors can easily add a "9/10 security rating" to their packaging without actually caring about

[1] `https://zmap.io/`
[2] `https://github.com/robertdavidgraham/masscan`
[3] `https://www.shodan.io/`

security.

Smart locks have a very large attack surface: many different components which are potentially vulnerable to attack. Manufacturers often add as many different systems as possible to make their system more competitive and "state-of-the-art". Many smart locks will use a combination of Wi-Fi, Bluetooth, RFID cards, fingerprint sensors, keypads, and mechanical backup locks. Each of these technologies has a combination of known attacks, outdated and vulnerable variants, and potential implementation issues. While security professionals are often knowledgeable on these problems, developers rarely have the same knowledge. This leads to insecure components being used in modern systems, as well as poor design choices which can lead to vulnerabilities in devices.

It is common for white-hat hackers to analyse the security of different devices to attempt to find methods of attacking the systems. Any exploits found can then be reported back to the manufacturer alongside advice on how to mitigate the issues, allowing them to improve the security of their system. This is often encouraged through bug-bounty programs, which reward researchers for reporting their findings instead of using their exploits maliciously. Furthermore, these analyses allow researchers to identify common issues for the creators of these devices, which is useful both for research purposes and also for advising developers on how to solve these problems when creating their own systems.

In this dissertation, I will analyse the security of two smart locks to identify common security problems faced by the IoT industry. I will perform a vulnerability analysis of the two systems, using both black box and clear box analysis techniques to uncover software vulnerabilities while also identifying other attacks on the systems. I will then analyse the set of exploits found to identify any common issues with the systems which may affect the industry as a whole, and suggest some approaches to solving these problems.

1.1 Choice of Locks

There are a wide range of locks that are considered "smart". Some simply provide one electronic access mechanism such as a fingerprint sensor or RFID unlock and claim to be smart, while others will provide an app to control the lock or use a wide range of electronic mechanisms. For my project, I selected modern locks which provide as much functionality as possible, and which either had not yet been analysed or had a very limited set of known attacks. I also prioritised popular locks such as

Amazon best sellers and systems which have been widely recommended.

The first lock selected is the Pineworld smart lock. This system uses passcodes, fingerprints, RFID smart cards, a mechanical backup lock, and the Tuya mobile app which controls the lock over Wi-Fi. This is a very popular lock that is recommended by multiple websites and has only been attacked once before (by PenTestPartners [1]). It was one of the most popular locks on Amazon at time of purchase (December 2020) and continues to be a popular product.

The second lock selected is a Ruveno smart lock. This provides passcode, fingerprint, and smart card credentials, as well as a mechanical backup lock. It uses the TTLock app to configure and control the lock using Bluetooth Low-Energy. It was the Amazon best-seller at time of purchase (February 2021) and uses an app that claims to be used by over 100 companies. The "Ruveno" lock is manufactured by Sciener, which appear to be the same company as TTLock. This implies that many of these companies are simply selling re-branded Sciener systems, and therefore it is very likely that vulnerabilities found in this lock will impact all of their 100+ brands.

1.2 Related Work

Several security researchers have previously analysed various smart locks to find vulnerabilities in their design. Rose and Ramsey [2] analysed a series of 16 Bluetooth Low-Energy based smart locks and found 12 of them to be vulnerable to an assortment of attacks, many of which are simple to perform. Several locks shared passwords in plaintext, others were vulnerable to replay attacks, and some were unlocked using bruteforce or fuzzing attacks. They also discussed common features of the locks they were not able to attack, providing useful starting points for other lock manufacturers.

Higher security systems have also been attacked: at Defcon 24, Jmaxxz [3] presented a range of attacks on the high-security August smart lock, which preceded separate analyses by Ye et al. [4] and Fuller et al. [5]. Jmaxxz showed that the debug menu could be accessed and exploited, the API leaked information, guests could become admins and avoid other guest restrictions, and encryption keys were being insecurely transmitted and stored. After these vulnerabilities were patched, Ye et al. presented four attacks on the August smart lock: accessing the key from the handshake, revealing the owner's account information, discovering personal data about the user, and a denial-of-service attack. All of their attacks assume access to the user's previously rooted or jailbroken device, which is a very unlikely threat. Fuller et al. discuss four main attempted attacks on the lock: a password changing attack

(a) The Pineworld Smart Lock

(b) The Ruveno Smart Lock

Figure 1.1: The two selected smart locks with their smart cards and keys

which could lock the user out of their app; two methods of maintaining credentials after they have been revoked; an attempt at bypassing timed credentials; and an attempt at unlocking the lock by recording Bluetooth messages. Few attacks were successful, suggesting that the security of the lock had significantly improved since Jmaxxz's attacks.

Other research groups have created blog posts detailing attacks on smart locks, such as PenTestPartners' analysis of the Tapplock [6] and Felch's analysis of the KeyWe smart lock [7]. The Tapplock used no encryption, was vulnerable to replay attacks, used weak key generation for pairing, and has a thin, easy-to-cut shackle. The KeyWe lock used weak key generation and was also vulnerable to replay attacks.

PenTestPartners have also published a blog post on the Pineworld smart lock [1], which is one of the two locks that I will be analysing. They detail a drilling attack on the lock, taking advantage of weak build materials to access the mechanism and

open the lock. They also briefly discuss other problems with the lock, although none are proper exploits.

Smart locks often use RFID cards, Bluetooth, and Wi-Fi as part of their system. Each of these technologies has its own history of vulnerabilities and current known exploits, discussed in detail in Section 2.1.

The most popular RFID standard is Mifare. The original Mifare Classic was reverse engineered by Nohl et al. [8] and then attacked by Garcia et al. [9], Gans et al. [10] and Garcia et al. [11]. The hardened version of Mifare Classic was later attacked by Meijer and Verdult [12]. Several versions of the more secure Mifare DESFire cards are also vulnerable: Oswald and Paar [13] found side-channel attacks on the original DESFire, and Hurley-Smith and Hernandez-Castro [14] and Flynn [15] found separate vulnerabilities in DESFire EV1.

Bluetooth has many versions with poor security, analysed in detail by Padgette et al. [16]. The pairing protocol has been attacked by Jakobsson and Wetzel [17], Wong and Stajano [18] and Antonioli et al. [19]. There is no encryption in Bluetooth versions before 4.0, and there are various known attacks on the protocol such as Bluesmacking, Bluejacking, and Bluesnarfing. Bluetooth Low-Energy (BLE) also has known issues, such as providing no user-level authentication or end-to-end security. Both Bluetooth and BLE provide various security levels and modes, several of which are vulnerable.

Wi-Fi has multiple encryption schemes, many of which have been attacked. Wired equivalent privacy has been attacked by Stubblefield et al. [20], Tews [21], and Tews and Beck [22], and is no longer used due to these vulnerabilities. There are several known attacks on WPA such as de-auth attacks, explored by Kraft and Holston [23], and the evil twin attack. Vanhoef and Piessens [24] discovered the KRACK attack on WPA2, and Vanhoef and Ronen [25] found a set of vulnerabilities dubbed “Dragonblood” in the proposed version of WPA3.

Chapter 2

Background

In this chapter, I explore the known exploits against technologies used by smart locks. I then design a threat model for smart locks, informed by the known exploits on the technologies in use. I also discuss the relevant laws and ethical concerns for security analyses, including computer misuse laws, the laws concerning reverse engineering and decompilation, and the ethics of vulnerability disclosure.

2.1 Technologies in use

Smart locks use multiple technologies as methods of controlling or opening the lock. Each of these components has multiple versions which may be insecure and several known vulnerabilities that impact the thread model. The main technologies used by the smart locks selected are RFID cards, Wi-Fi, and Bluetooth Low-Energy. In this section, I explore the different versions of each of these components, as well as known security problems with them.

2.1.1 RFID Access Control

Many smart locks use RFID tags or smart cards as a method to operate the lock. A majority of these systems are vulnerable to attacks, making RFID credentials a target for hackers. RFID access control can be broadly grouped into two systems: low-frequency (LF) tags and high-frequency (HF) cards. LF tags usually function by sending a constant string from the tag's memory to the reader, which can be cloned onto an attacker's card or replayed to the reader at a later time. While some HF protocols such as PROX use a similar system to the LF tags, there are more complex and secure HF systems available such as the Mifare family of RFID cards.

These split memory into sectors, each of which is protected by two keys that should only be known to card readers and writers. More complex access control can then be used with these cards.

The most common attack on HF smart card systems is card cloning. Assuming that the attacker knows the keys for each sector of the card (or can learn them through vulnerabilities in the system), they can read all data from the card and make a perfect copy. Mifare attempts to prevent this attack by making sector 0 read-only, preventing a legitimate card's UID from being overwritten. Attackers can use cards with a "Chinese magic backdoor" to allow them to write to sector 0, or use devices such as the Proxmark[1] or the Chameleon[2] to emulate an arbitrary HF-RFID card.

The design of Mifare Classic was kept secret by NXP to provide increased security by obscurity. Nohl et al. [8] reverse-engineered the protocol and found several attacks against it. Garcia et al. [9] found further attacks on the protocol before de Koning Gans et al. [10] published a more practical attack on the protocol. Garcia et al. [11] demonstrated a series of further attacks, the most impactful of which is the "nested" attack which is now used by devices such as the Proxmark[1] to crack any unknown access keys on Mifare Classic cards in mere seconds. A "hardened" version of the cards was later released, however Meijer and Verdult [12] found attacks against the Crypto-1 cipher in 2015 that compromise all versions of Mifare Classic. The "hardnested" attack was added to the Proxmark to allow easy exploitation, although it takes significantly longer than the nested attack.

Other variants of Mifare have also been found vulnerable to attacks. DESfire cards were found to be vulnerable to side-channel attacks by Oswald and Paar [13], and research by Hurley-Smith and Hernandez-Castro [14] and Flynn [15] shows that DESFire EV1 is also vulnerable. Several standards are still considered secure: DESfire EV2 and EV3 have no known attacks and Mifare Plus is considered secure when operating at the highest security level.

These smart cards must be configured such that an attacker cannot obtain the data required to clone the card. For example, if a reader only inspects the world-readable UID from the card, then anyone can read and copy this UID to gain access. Similarly, only using default keys could allow an attacker to clone the card by using a dictionary attack.

[1] `https://proxmark.com/`

[2] `https://github.com/emsec/ChameleonMini`

2.1.2 Bluetooth

Bluetooth is a protocol that allows pairs of devices to communicate with each other over short distances. Bluetooth Low-Energy (BLE) is a variant that uses less power and is ideal for devices that do not send large quantities of data. The Bluetooth protocol has a poor history of security: there was no encryption in the protocol until version 4.0; attacks on the pairing protocol have been found by Jakobsson and Wetzel [17], Wong and Stajano [18], and Antonioli et al. [19]; and there are multiple known attacks which are possible on modern versions of both protocols.

Bluetooth has a variety of known attacks. Bluesmacking is a denial-of-service attack on Bluetooth which sends an oversized ping packet to the device, causing it to crash. Bluejacking allows any user in range to send unsolicited messages to a device, and Bluesnarfing allows attackers to read information such as a user's contacts. Bluetooth devices operate at a specified security mode and level; low security levels and modes are vulnerable to an assortment of attacks. Padgette et al. [16] published a survey of all known Bluetooth attacks in all versions in 2017, providing comprehensive coverage of attacks and best practices for Bluetooth.

BLE also has multiple known security issues. The protocol provides no user-level authentication, enabling impersonation and man-in-the-middle attacks. There is no end-to-end encryption, which enables traffic eavesdropping and modification. These mean that application developers must implement encryption, message authentication, and user authentication to prevent these attacks. BLE also has configurable security modes and levels, several of which are insecure.

2.1.3 Wi-Fi

There exist a range of Wi-Fi encryption schemes, most of which are vulnerable to attack. Wired equivalent privacy (WEP) was introduced in 1997 and replaced by Wi-Fi protected access (WPA) in 2003. WPA2 was ratified in 2004 and is now the most widely used Wi-Fi security mechanism. WPA3 was announced in 2018, however it was only certified in 2020 after vulnerabilities were found in the original proposal. As WPA3 was only recently certified, the majority of systems currently use WPA2.

WPA2 has several well-known attacks. The de-auth attack [23] sends a packet to devices connected to a Wi-Fi network which forces them to disconnect from the network. Evil twin attacks involve duplicating an existing network to perform a man-in-the-middle attack. Vanhoef and Piessens [24] discovered the KRACK attack

on WPA2 which forced the encryption key to be re-used, allowing messages to be decrypted, replayed, or forged.

WPA and WPA2 are both vulnerable to password cracking attacks. When a device connects, it uses a four-way handshake to exchange setup information. This includes a hash of the network password, which can be cracked with offline brute force and dictionary attacks. De-auth attacks can be used to disconnect devices from the network, allowing handshakes to be captured when the device immediately attempts to reconnect.

Vanhoef and Ronen [25] found a set of five "Dragonblood" vulnerabilities in the proposed WPA3 standard. These include a downgrade attack that enables WPA2 attacks to be used, a security group downgrade attack, two side-channel attacks which leak the network password, and a denial-of-service attack. Although these were patched before release, the same researchers later found a pair of vulnerabilities stemming from the imperfect handling of the Dragonblood vulnerabilities. These are present in the first deployed versions of WPA3 but are patched in modern versions.

2.2 Threat Model

A threat model identifies the important threats to a system, allowing security analysis to focus on the most relevant issues. This includes the threat actors that may attack the system, the attack scenario and any assumptions that can be made, and the possible attacks that are relevant to the system.

2.2.1 Threat Actors

The first threat actors are skilled technical attackers ("hackers"). They are assumed to understand all components of a system; by Kerckhoff's principle, the system should be secure even if the attacker knows everything about it except for the private keys. Furthermore, skilled attackers can create their own exploits and are not constrained to widely-available tools. While this is the threat with the highest potential, they are also the least common, and may not be a priority.

The second threat actors are non-skilled technical attackers ("script kiddies"). This attacker is unlikely to understand how the system works or to have the capability to write their own exploits, instead of relying on widely available tools such as those in Kali Linux[3]. Script kiddies are more common than hackers, but not nearly as

[3] `https://www.kali.org/`

competent.

The third type of attacker has physical access to the lock. They will attempt attacks on the lock mechanism and may attempt to access credentials such as fingerprints or passcodes to open the lock. This attacker is unlikely to possess any technical skills and will focus their attacks on the physical aspects of the lock.

The final threat actors are guests. This attacker is given legitimate access to the system for a limited time using mechanisms such as remote entry requests or temporary credentials, and will attempt to attack the lock at a later date. The primary threat is privilege escalation: using their access to either get access to another user's credentials or extending their access past the assigned access period. Their skills may overlap with any of the prior attacker models.

2.2.2 Attack Scenario

Multiple assumptions can be made about the attack scenario based on the known attacks on different technologies. These include the times when the attacker has access to the lock, their ability to perform certain attacks and the ease of the user detecting an attack in progress.

The attackers will have access to the lock at different times which impacts the attacks they can perform. Attackers may be near to the lock when it is first being set up, or at a later point in time when the lock is in use. An attacker can return to the lock at a later time to attack it; this will allow them to perform attacks undetected if they would otherwise have been noticed by the user. A guest user can more easily get access to the lock due to their legitimate temporary access.

Man-in-the-middle (MITM) attacks are possible on both Wi-Fi and Bluetooth. The Bluetooth MITM attack can be performed without the user noticing, however the Wi-Fi MITM attack will interrupt their current connection and relies on the user attempting to connect to the malicious network manually. Passive MITM attacks can be performed by either technical attacker, however active MITM attacks that modify traffic are not possible for non-skilled attackers.

2.2.3 Potential Attacks

Opening the lock An attacker may attempt to open the lock through various methods. They could learn a passcode for the lock, copy a fingerprint, clone an RFID card, or register new credentials to gain access. They may also attempt to unlock the system over Bluetooth or Wi-Fi, either by directly connecting to the

device or by using the man-in-the-middle attacks possible with either technology. Furthermore, an attacker could attempt to reset the system and control it with their own devices. Finally, an attacker could attack the mechanical backup lock.

Privilege Escalation An attacker who is given some level of access to a system may attempt to increase their privileges beyond their intended access. Many smart locks provide guest access to a system, which may include time-constrained access to the lock. The guest could attempt to extend their access or increase the operations they can perform on the lock. They may also attempt to escalate from a guest to a normal user or an administrator if the lock has separate access levels for users.

Denial-of-Service (DoS) An attacker may attempt to modify or remove existing credentials for the system to prevent the user from opening the lock. Both skilled and non-skilled attackers can use known DoS attacks on wireless technologies, such as the de-auth attacks on Wi-Fi or Bluesmacking on Bluetooth. An attacker may be able to reset the lock, removing the existing configuration, and take control of the system themselves. Finally, an attacker may be able to damage or remove exposed components from the lock to prevent their use.

Information Leakage It is unlikely for there to be many attack vectors under which an attacker can leak information from a smart lock, however information leakage must still be prevented. One possibility is for an attacker to obtain credentials used by the lock if they are improperly stored, shared, or generated. This concerns Wi-Fi passwords, API keys, account credentials, dynamic passcodes for the lock, and guest access tokens, which the lock must process carefully.

2.3 Law and Ethics

Vulnerability analysis is a legal and ethical grey area for a variety of reasons. Hacking into systems without authorisation is illegal around the world; in the UK, the Computer Misuse Act 1990 [26] prohibits this. Some methods used during analysis are also subject to laws: decompilation and reverse engineering potentially infringe on copyright and must be used in a restricted manner. Some companies will run bug bounty programs for researchers to report vulnerabilities, however others do not and vulnerability disclosure in these cases requires caution.

2.3.1 The Computer Misuse Act

In the United Kingdom, the Computer Misuse Act 1990 [26] is the primary law for computer security offences, containing five sections on computer misuse offences. This project must avoid violating sections 1, 2, 3, and 3A. Section 3ZA concerns "unauthorised acts causing, or creating risk of, serious damage", which does not apply to this project.

Section 1 prohibits unauthorised access to computer material. Article 1 states:

> A person is guilty of an offence if -
>
> (a) he causes a computer to perform any function with intent to secure access to any program or data held in any computer or to enable any such access to be secured;
>
> (b) the access he intends to secure, or to enable to be secured, is unauthorised; and
>
> (c) he knows at the time when he causes the computer to perform the function that that is the case.

My dissertation will attempt to find vulnerabilities in a series of smart locks. These vulnerabilities may be exploited in a manner that provides (or enables) access to data on the locks, which satisfies section 1 article 1a. To keep my project compliant with the Computer Misuse Act 1990, I must ensure that the access I gain is authorised — for information about myself or a lock that I own, access is authorised by myself. For any computer that is not my own, such as cloud servers or web servers, I do not have authorised access and will not attempt to attack these systems.

Section 2 concerns unauthorised access with intent to commit or facilitate commission of further offences. The exploits created through this project could theoretically be used to open a lock and enable trespass, theft, or other crimes. If no offence is committed under section 1 of the act, or if there is no intent to commit or facilitation other offences, then this section does not apply. Through my compliance with section 1 of the act and my intent being to identify and report any vulnerabilities found, this project will not violate section 2 of the act.

Section 3 covers denial-of-service attacks and damage to systems that prevents them from functioning as intended. Denial-of-service attacks are part of the threat model for this project, and must only be performed with authorisation. As with other sections of the act, this means I must only attack my own system, and avoid interfering with any system I do not have permission to test on.

Section 3A concerns "making, supplying or obtaining articles for use in offence under section 1, 3 or 3ZA". During this project, I will create tools that are capable of exploiting the vulnerabilities found, as "proof-of-concept" exploits. These tools will be used solely to prove the presence and impact of vulnerabilities. They may be shared with the system manufacturers to demonstrate my findings, but will not be shared with any other third party.

2.3.2 Decompilation and Reverse Engineering

Reverse engineering of software allows vulnerability researchers to understand how a system has been implemented and to identify vulnerabilities in the code. Reverse engineering is an ethical grey area: although it is beneficial to researchers, it can be used to learn about protected intellectual property or to infringe on copyright. Additionally, decompilation is often used to convert compiled software into a higher-level language for analysis, which presents its own legal and ethical challenges.

The main legislation concerning reverse engineering in the United Kingdom is the Copyright, Designs and Patents Act 1988 [27] following an amendment by The Copyright and Related Rights Regulations 2003 [28]. Section 50B article 1 of the act states:

> It is not an infringement of copyright for a lawful user of a copy of a computer program expressed in a low level language—
>
> (a) to convert it into a version expressed in a higher level language, or
>
> (b) incidentally in the course of so converting the program, to copy it,(that is, to "decompile" it), provided that the conditions in subsection (2) are met.

Section 50BA article 1 states:

> Observing, studying and testing of computer programs
>
> (1) It is not an infringement of copyright for a lawful user of a copy of a computer program to observe, study or test the functioning of the program in order to determine the ideas and principles which underlie any element of the program if he does so while performing any of the acts of loading, displaying, running, transmitting or storing the program which he is entitled to do.

This means that it is permitted to decompile and reverse engineer the program for the purpose of testing its security, which I will do during this project.

2.3.3 Bug Bounties and Vulnerability Disclosure

After vulnerabilities have been identified in a system, they should be reported to the manufacturer so that they can be patched. A common approach to encourage hackers to report these exploits is to operate a bug bounty program. This allows issues identified to be reported to the company privately, providing a reasonable amount of time to mitigate the vulnerability, while also rewarding those who report these issues.

Bug bounties define a specific scope for hackers. This sets out which parts of the system can be attacked, and which must be left alone. It is important that white-hat hackers learn about and carefully follow the scope provided by the company to prevent any violation of the Computer Misuse Act 1990 [26].

Not all companies offer a bug bounty program or other vulnerability disclosure scheme. In these cases, attacking their systems is not authorised, and must be avoided. Any vulnerabilities identified in their systems should still be reported to prevent their malicious exploitation, however great care must be taken to research these issues in a legal manner.

The vulnerabilities identified in this project will be reported to the manufacturers after the project is completed. It is unlikely that the vulnerabilities will be reported during the dissertation due to time constraints. The reports will include details on the underlying issues, how they can be exploited, and how the vulnerabilities can be prevented. I will continue communicating with the manufacturers after my report to resolve any queries if required.

Chapter 3

Methods

Vulnerability analysis methods can be split into black box and clear box analysis. Black box analysis consists of methods where the system is treated as a black box that receives some input, processes it in a way that cannot be observed, and provides some output. The inputs can be either observed or manipulated by the researcher, and the corresponding outputs can be used to infer potential attacks. There is also a variety of useful information that can be observed by scanning the system, and several attacks that can be performed without any understanding of the underlying implementations. Clear box analysis involves reverse engineering the system to gain an understanding of how the system functions, uncovering issues in its implementation. This uses reverse engineering techniques, often requiring the use of decompilers and related tools.

3.1 Black Box Analysis Methods

Black box testing allows some vulnerabilities in the system to be identified without needing to understand the system. These attacks primarily consist of observing the system and its behaviour through traffic monitoring and fuzzing, as well as attempting to find exposed attack surfaces through enumeration. The exposed services may be able to be attacked at this stage.

3.1.1 Network Analysis

A large amount of information can potentially be gained by analysing network traffic over both Bluetooth and Wi-Fi. Analysing the traffic can reveal issues in the system which can then be exploited.

Monitoring Traffic

Network traffic can be captured using tools such as Wireshark or tcpdump on a device capable of running them. To intercept traffic on other devices, the researcher can either run a man-in-the-middle to observe device traffic or they can use features of the device itself. For example, Android phones can record Bluetooth traffic through the "HCI snoop log" available in developer mode, and record Wi-Fi traffic by using an app which acts as a VPN for the phone.

The data captured are then analysed to identify the clients communicating, the types of packets, and to eavesdrop on data in transit. This data may be encrypted, however any unencrypted or improperly encrypted data can be monitored. This gives insight into possible attacks such as replay attacks or traffic manipulation with man-in-the-middle attacks.

Replay Attacks

In a replay attack, a message or sequence of messages is recorded and later replayed to the device in an attempt to repeat the operation. For smart locks, this might be a message to unlock, add a new credential, or change configuration settings. Replay attacks are possible when there is no token of "freshness" in a message, which would prevent the message from being reused later. If there are messages in the recorded traffic that are identical to other messages, this indicates that a replay attack is possible. Replay attacks may also be possible when this is not the case — for example, if a packet is AES-CBC encrypted with a random IV each time, then the encrypted message will change every time; however, if there is no check for a repeated IV and the plaintext messages are identical, then messages can be reused even though they never repeat in the monitored traffic.

Man-in-the-middle Attacks

A man-in-the-middle (MITM) attack puts an attacker between two devices that are communicating, sending messages to each device while pretending to be the other. The attacker can then observe all traffic and modify messages, inject new packets, or prevent data from being delivered.

MITM attacks can be passive or active. In a passive MITM attack, the attacker receives messages from each device and forwards them without interference. The attacker can then eavesdrop on all messages between the devices for analysis. An active MITM attack will interfere with the traffic being sent between the devices.

This may be used to prevent some messages from being delivered, send a new message while impersonating a device, or modify the data in transit.

MITM attacks are possible on both Wi-Fi and Bluetooth. An attacker can set up an "evil twin" network for Wi-Fi that impersonates an existing network. If WPA or WPA2 are being used, then a de-auth attack can be used to disconnect devices from the legitimate network, potentially connecting to the malicious network when they attempt to reconnect. This attack can be performed with the Aircrack-ng[1] suite of tools or automated with scripts such as Airgeddon[2]. Bluetooth provides little MITM protection and is known to be vulnerable to the attack. Tools such as BTProxy[3] can be used to MITM a Bluetooth connection, and Gattacker[4] can be used to attack BLE devices.

3.1.2 Fuzzing

One method of testing the system is to provide large amounts of modified data to the system and observing the result, known as fuzzing. By procedurally modifying each part of a command and then sending the packet, a researcher can identify which parts of the packet cause the system to change its behaviour and which have no effect. Fuzzers may cause errors when certain data is malformed, which may produce useful behaviour or demonstrate the presence of another vulnerability which can then be exploited further.

In the case of smart locks, a common safety mechanism is for a lock to unlock as soon as it enters an error state (preventing anyone from being locked into a burning building, for example). This benefits an attacker who wishes to open the lock, as errors bypass the security mechanisms of the system. Malformed data may also cause the lock to crash entirely, leading to a denial-of-service attack.

Although fuzzing is intended as a black box technique, a fuzzer can benefit from clear box analysis. For example, clear box analysis can reveal the structure of custom packets, which provides information about the intended purpose of fields and their expected values. Furthermore, if clear box analysis reveals data such as encryption keys, then the results from each fuzzed packet can be decrypted to reveal further information about the system's behaviour in response to the modifications.

[1] `https://www.aircrack-ng.org/`
[2] `https://github.com/v1s1t0r1sh3r3/airgeddon`
[3] `https://github.com/conorpp/btproxy`
[4] `https://github.com/securing/gattacker`

3.1.3 Service Enumeration

Enumeration over the Internet

Internet-connected devices expose services on specified ports to allow other devices to communicate with them. Network scanning tools such as Nmap[5] can be used to identify open ports using a wide range of scanning methods, often identifying the services on each port and potentially discovering the version of the service. These scans indicate services that are running vulnerable versions or which may be exploitable by an attacker.

Enumeration over BLE

BLE devices advertise services using Generic ATTribute (GATT) profiles. These services can have multiple characteristics, each accessed at a specified handle with a public UID. Some of these characteristics are static values such as the manufacturer name and model number which can give some insight into the device. There are also a range of standardised services that can be identified, and which may have known vulnerabilities that an attacker can exploit. There are a range of BLE enumeration tools, including Bettercap[6] and GATTTool (from the BlueZ[7] suite).

3.1.4 RFID Hacking

RFID smart cards can be attacked without any reverse engineering of the system. I will be using the Iceman fork of the Proxmark3[8] to attack the smart cards, and the Chameleon Mini[9] to clone the cards.

RFID cards can be attacked in four stages. First, the type of card is identified, as well as additional information such as the type of PRNG to indicate the attacks possible on the card. Then, any default keys can be identified using a dictionary attack. Known attacks such as the nested attack are then used to crack any unknown keys. Finally, the data on the card is dumped and used to clone the smart card.

An alternative approach is to use the "autopwn" command to automate the entire process. This makes RFID hacking accessible to script kiddies as well as experienced hackers.

[5] `https://nmap.org/`
[6] `https://www.bettercap.org/`
[7] `http://www.bluez.org/`
[8] `https://github.com/RfidResearchGroup/proxmark3`
[9] `https://github.com/RfidResearchGroup/ChameleonMini`

3.2 Clear box analysis methods

Clear box (or "white box") analysis involves reverse engineering a system to understand how it functions. By understanding the protocols involved, vulnerabilities in the design and implementation can be identified and used to attack the system. While binary code or other compiled representations of programs can be reverse engineered directly, it is beneficial to "decompile" the program into a higher level language. The result is much closer to the source code — in the best case scenario, everything except variable names can be reconstructed. This allows for the high level functionality of the system to be understood, and some potential low level vulnerabilities to be identified. Both decompilation and reverse engineering are a legal and ethical grey area, which is discussed in detail in Section 2.3.2.

3.2.1 Decompilation

The objective of decompilation is to take code that has been compiled into a low-level representation (such as binary, Java bytecode, or an Android APK) and convert it back into a higher level representation. Binary files can be disassembled to obtain assembly code, which can then be analysed to reconstruct high-level functionality. Bytecode and APKs can be similarly analysed, however they will be reconstructed into object-oriented code containing classes, modules, and libraries in a large collection of files. The resulting high-level code is far easier to reverse engineer than the low-level representation.

Tools

For binary decompilation, there are two main tools: Ghidra[10] and IDA Pro[11]. Ghidra is a free, open-source tool created by the NSA which was released to the public in 2019. IDA Pro is a paid decompiler created by Hex Rays that advertises itself for malware analysis and security research, which also provides an evaluation version with limited functionality. Both of these decompilers provide a full suite of reverse engineering tools in addition to a decompiler, however Ghidra is free while IDA Pro requires a license for the full version. I therefore used Ghidra for this project.

There are also multiple tools for decompiling Android APKs. Decompilers such as dex2jar[12] and APKTool[13] can be used from the command line, while websites

[10] `https://ghidra-sre.org/`
[11] `https://www.hex-rays.com/ida-pro/`
[12] `https://github.com/pxb1988/dex2jar`
[13] `https://ibotpeaches.github.io/Apktool/`

such as APK decompilers[14] automate the process for the user. These compilers have varying decompilation quality, with some systems failing to decompile certain functions completely. I combined the output of multiple decompilers to get a complete decompilation of the applications I reverse engineered.

Improving Decompilation Quality

Decompilation is an imperfect process — while class and function names may be persevered if debugging symbols are included, it is more common for the decompilation to result in source code with no useful names. Furthermore, data types are unlikely to be reconstructed correctly, as there are many cases where a compiled struct, array, or list of variables cannot be distinguished from other types. The code may also have been obfuscated in an attempt to prevent reverse engineering, which will make the resulting code much harder to interpret.

Some decompilers and IDEs will contain additional reverse engineering tools to improve the quality of the decompilation manually. These tools allow automatic refactoring, including changing variable names or types and creating custom data types. Additionally, there are tools and plugins which can be used to help identify known parts of binaries automatically. For example, the FindCrypt[15] plugin for Ghidra will scan a compiled binary and automatically identify cryptographic constants, allowing rapid identification of cryptographic functions in the decompiled binary. Furthermore, de-obfuscation tools such as DeGuard[16] and Deoptfuscator[17] can attempt to undo certain obfuscation techniques for Android applications, making their source code easier to interpret.

3.2.2 Reverse Engineering

Reverse engineering is not a well-defined process; instead, it is largely up to the researcher to identify how the system functions from the reconstructed source code. It is useful to begin by identifying core functionality that needs to be implemented securely, which will largely depend on the threat model. It is also useful to explore libraries and their usage, especially if there are custom libraries in use for security features such as cryptography.

The threat model in this project indicates the parts of the source code that could lead

[14] `https://www.apkdecompilers.com/`
[15] `https://github.com/d3v1l401/FindCrypt-Ghidra`
[16] `http://apk-deguard.com/`
[17] `https://github.com/Gyoonus/deoptfuscator`

to exploits, and are therefore important functions to reverse engineer. An attacker may be able to unlock the lock if the unlock function is poorly designed, or if the credential management is not secure. Privilege escalation prevention also depends on credential management, in addition to the handling of different user levels such as guests and administrators. Poor error handling could lead to denial-of-service attacks, as well as anything which can block other devices that attempt to interact with the lock. Finally, any functionality which stores, shares, or generates secret information can be reverse-engineered to identify potential information leaks.

While there may be multiple types of library that are interesting to reverse engineer, one of the most likely places to find implementation errors are cryptography libraries and their usage. Although many companies advertise "state-of-the-art military-grade encryption", this does not imply their system is secure — it ignores many parts of their security and there are many ways that their military-grade encryption can be misused. Important things to identify and examine include key exchange, the encryption function used and its parameters, generation of initialisation vectors and nonces, message authentication and signing algorithms, and their (pseudo) random number generators.

Chapter 4

Vulnerabilities, Exploits, and Countermeasures

In this chapter, I will discuss the vulnerabilities in each component of both systems analysed, methods for exploiting these design flaws, and the countermeasures that the lock designers can implement to mitigate the vulnerabilities. Section 4.1 explores the RFID smart card systems used, Section 4.2 discusses physical security components including the backup locks and fingerprint sensors, Section 4.3 covers an attack on the Pineworld lock's Wi-Fi setup, and Section 4.4 details multiple attacks on the Ruveno/Sciener lock's Bluetooth Low-Energy (BLE) protocol using the TTLock app.

4.1 RFID

RFID security depends on two factors: the technology in use and the card configuration for the system. I have attacked both smart locks through their choice of the vulnerable Mifare Classic standard, and one has a configuration that is vulnerable to a separate attack.

4.1.1 Pineworld

The Pineworld smart cards are configured to use the access key `0xFFFFFFFFFFFF` for all sectors except sector 15, which uses `0x000000000000` as the B key and a random unknown value from the lock as the A key. Sector 15 can be read with the all-0 key, which contains a copy of the 4 byte UID and 12 more unknown bytes. Experimenting with the data on the card shows that the lock ignores these unknown

bytes and instead only reads the copy of the UID to check that it is the same as the card's UID. The lock reads this sector with the secret key, preventing an attacker from cloning the card based on the UID alone.

The smart lock uses Mifare Classic with a weak PRNG. This means that it is vulnerable to the nested attack, and the unknown access key (sector 15's A key) can be cracked rapidly with access to the smart card. The card can then be cloned and used to open the lock.

4.1.2 Ruveno

The Ruveno lock's smart cards are also using Mifare Classic with a weak PRNG. While this would make it vulnerable to the nested attack, this is not necessary: the card is blank apart from sector 0, which implies that the lock is only testing for a known UID. Further testing confirms that the 4-byte UID is the only part of the card which needs to be cloned to open the lock, which is advertised by the smart card. This configuration allows the card to be cloned after having only instantaneous access to the card.

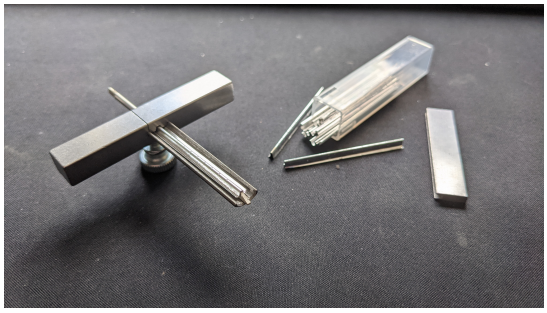
4.1.3 Countermeasures

Both smart card systems need to be updated to a more secure version of Mifare such as DESFire EV2 or EV3. Additionally, they should be using a more secure configuration, such as having a shared secret between the lock and smart cards stored in a sector that is protected by two secret access keys, rather than having an easily guessable key for users to read the data from the card.

4.2 Physical Security

There are several physical elements to the locks that need to be secured. In this section, I discuss the security of the mechanical backup locks employed by both systems, as well as attacks on the fingerprint sensors, including a design flaw in the Pineworld lock.

There already exists one known physical security flaw in the Pineworld lock, identified by PenTestPartners [1]. In their attack, they note that the weak material choice allows an attacker to drill into the side of the lock and manually engage the unlocking mechanism. This can be fixed by using a stronger material for the lock body.



(a) Equipment for a foiling attack

(b) My impressioned foil key compared to the provided key

Figure 4.1: Foil impressioning attack demonstration

4.2.1 Backup Locks

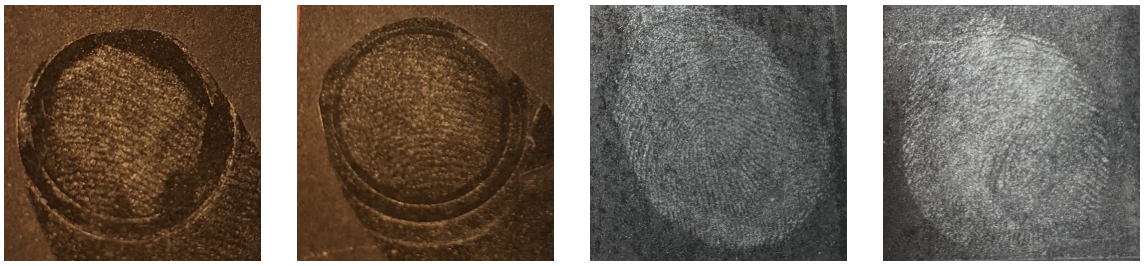
The Pineworld smart lock's mechanical lock is relatively secure. It is a slider lock with 18 sliders, which is far more than normal — the most common use of slider locks is for cars, which usually have between 6 and 10 sliders. Although these sliders have few possible positions each, it is still a difficult lock to open.

The Ruveno smart lock has several questionable choices with its backup lock. It uses a dimple lock which can be picked or raked open, and it is also vulnerable to the "foiling" attack — a form of impressioning which allows an attacker to make a key from a piece of tinfoil and a cheap foiling kit using only the lock, as shown in figure 4.1. The key appears to be for a 6-pin dimple lock, however disassembling the lock reveals that it is only a 4-pin lock (shown in figure 4.2), which makes it far easier to attack. The small lock is forced due to the restrictive choice to have the mechanical lock on the front of the lock, limiting the depth of the lock. A better design would use a higher security lock with more pins in a position that allows for a larger core, such as on the bottom of the lock. Alternatively, using a cross lock would allow for more pins to be added without moving the lock.

Figure 4.2: My disassembly of the Ruveno mechanical lock

4.2.2 Fingerprint readers

A known attack on fingerprint readers is the ability to clone a fingerprint and to use the clone to unlock the lock. Researchers at Talos [29] achieved an 80% success rate with fake fingerprints, although the process is described as "difficult and tedious work". Fingerprint cloning is possible if there are any "latent" fingerprints on the lock, which can be more easily pulled off of certain surfaces such as glass. For both locks, I have successfully obtained a fingerprint from both the fingerprint sensor itself and from the passcode entry which could be used to clone the fingerprint, as shown in figure 4.3.



(a) From Ruveno fingerprint sensor

(b) From Pineworld fingerprint sensor

(c) From Ruveno passcode entry

(d) From Pineworld passcode entry

Figure 4.3: Fingerprints obtained from the sensor and passcode entry of each lock

An additional security issue specific to the Pineworld smart lock is the ability to access the fingerprint sensor hardware. The sensor is placed inside of the lock handle which can be disassembled from the outside using two screws on the back of the handle with a right-angled screwdriver, shown in figure 4.4a. Accessing the sensor allows for a variety of denial-of-service attacks on the sensor, including cutting the ribbon cable (figure 4.4c) and removing the circuit board or the sensor (figure 4.4d). While it is convenient to have the sensor on the handle for usability, it would be more secure to place it on the lock body so that it cannot be attacked once the lock is installed.

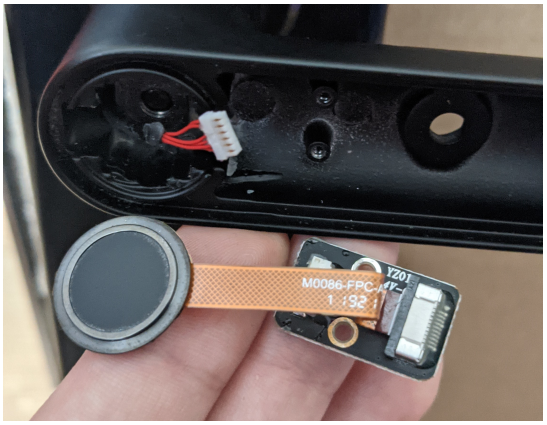
4.3 Wi-Fi Attacks (Pineworld / Tuya Smart)

The Pineworld lock, which uses the Tuya Smart app, is controlled over Wi-Fi. Connecting the lock to a Wi-Fi network requires a method of passing the Wi-Fi credentials to the lock. In this scenario, the primary concern is information leakage, as the Wi-Fi password should remain secret. Once the lock is connected to Wi-Fi, all communications are between the app and a set of cloud servers which in turn connect back to the lock, all protected by TLS. This means that the only part of

(a) Disassembling the handle while the lock is installed

(b) The exposed sensor

(c) Attacking the sensor with scissors



(d) Removing the fingerprint module

Figure 4.4: Attacking the exposed fingerprint sensor

their Wi-Fi system that I can attack is the setup process, as servers are out of scope.

4.3.1 Wi-Fi Setup Information Leak

The Wi-Fi setup has four steps:

- input the Wi-Fi credentials to the app
- put the lock into Wi-Fi connection mode
- start the app connection process
- the lock connects itself to the Wi-Fi network and communicates with a cloud server

During the third stage, the phone will broadcast two series of packets repeatedly: the first series of packets are empty UDP packets sent to IP addresses of the form `216.N.b2.b1`, where N is a sequence number and b1b2 are two bytes of data; the

second series are UDP packets broadcast to `255.255.255.255` which have a varying number of null bytes as data. The intention appears to be for the phone to set up a temporary direct Wi-Fi connection with the lock, which these packets can then be broadcast over and decoded, however the packets are transmitted on whichever network the phone is currently connected to.

As discussed in Section 2.1.3, it is possible to perform an "evil twin" attack to force a device to connect to an impersonated network. By performing an evil twin on a device that is setting up a smart lock, these two streams of data can be intercepted. Decoding the data from the first stream from the format above reveals descriptive information about the network, including the SSID (human-readable network name) and the BSSID (MAC address). To decode the second stream, take a list of the lengths of the null bytes, and remove any numbers under 256. Subtracting 256 from each remaining length and then converting to ASCII reveals the network password.

To prevent this information leakage, the information should not be broadcast until a direct connection between the two devices has been made. Alternatively, a more secure setup mechanism such as the temporary hotspot method supported by the app for other devices could be used.

4.4 Bluetooth Attacks (Ruveno / TTLock / Sciener)

Bluetooth is vulnerable to application-level man-in-the-middle attacks, and application developers need to implement appropriate protections against this threat. The TTLock app used by the Ruveno lock have used encryption for some of the data transmitted, however this has been done improperly at several levels. The key exchange between the lock and the app is not secure, there are a chain of design choices leading to a replay attack, and there are several other design choices that are not exploitable by themselves but could lead to attacks in the future.

All Bluetooth attacks found depend on a MITM attack.

4.4.1 Improper AES Key Exchange

When the lock is first connected to the lock, an AES key is generated by the lock and shared with the application after receiving the GET_AES_KEY message, which is used to encrypt all further communications. Both the get message and the reply encrypt data using the hardcoded AES key

```
0x987623e8a923a1bb3d9e7d0378124588
```

which is found as a byte array in the `ble.Command` class of the Android app. An attacker that uses a MITM attack during the setup process will record these two messages (as well as the rest of the setup process, which includes other sensitive information) and can decrypt the messages using the default key found.

After decrypting the AES key, all future communications can be decrypted. An attacker can immediately use this to decrypt the setup process, which includes information such as the admin key and unlock key used for the challenge-response protocol and the admin passcode (a backup keypad passcode which is hidden from the user unless they extensively explore the lock settings menu). Future messages can also be decrypted, revealing new passcodes added, the UID of new smart cards (which is sufficient to clone the card for this lock; see Section 4.1.2) and all other messages sent. An attacker can also encrypt their own messages, allowing them to interact with the lock as desired after sufficient reverse-engineering of the protocol. As a proof-of-concept, I created a script for an attacker to unlock the lock when desired.

To prevent this attack, a secure key exchange protocol should be used; ideally, this would be a standard variant of Diffie-Hellman key exchange which is known to be secure; elliptic curve Diffie-Hellman would be ideal as it uses much smaller values than regular Diffie-Hellman. A NIST standard curve should be used to prevent attacks on weak elliptic curves.

4.4.2 Replay Attack

The second discovered attack on the TTLock BLE protocol is a variant of a replay attack. This depends on a MITM attack which records at least one challenge-response exchange before an authenticated command; this is sent before every function except LOCK, UNLOCK, TIME_CALIBRATE, and GET_LOCK_TIME. A series of design choices allow the response to be replayed and an authenticated command to be performed, explored in turn below.

Static IV for AES-CBC

The encrypted part of the packets is encrypted under AES-128-CBC. CBC mode requires an initialisation vector (IV), which should never be reused. However, in this protocol the IV is the same as the encryption key, and therefore it is static. Re-using an IV means that the same plaintext will always encrypt to the same ciphertext, which reveals when a message uses the same value. In the protocol

discussed below, this allows an eavesdropper to identify that the CHECK ADMIN messages are always the same and therefore can be replayed.

Absence of replay protections

To prevent replay attacks, messages should include some data which makes the message invalid in the future. One possible approach is the use of a "nonce", or number only used once, which can be recorded by the lock. Any message which reuses a nonce can then be ignored, preventing the replay attack.

Alternatively, a timestamp can be included in the message so that it cannot be reused in the future. This approach is used for LOCK and UNLOCK, which prevents this attack — however, no other message includes this data.

A third possible approach is to use a challenge-response protocol which cannot be replayed due to constantly changing challenges. TTLock also implements this, but in an insecure manner which I attack below.

Weak Challenge-Response Protocol

The TTLock challenge-response protocol consists of four messages before a command can be sent:

App → Lock: CHECK ADMIN (admin key)
Lock → App: RESPONSE to CHECK ADMIN (success, 2 byte challenge)
App → Lock: CHECK RANDOM (sum(challenge, 4 byte unlock key))
Lock → App: RESPONSE to CHECK RANDOM (success)
App → Lock: COMMAND (data)

The first message contains the admin key exchange during setup, which is a constant value that can be replayed. The lock then replies with a challenge, which is only 2 bytes long. This means there are $2^{16} = 65536$ possible challenges, which is far too small to be secure. This means that an attacker can mass-replay the CHECK ADMIN message until a challenge is reused which the attacker already knows the response for.

The Attack

An attacker must run a passive MITM attack to record at least one full challenge-response exchange between the application and the smart lock. At a later time, the attacker can begin mass-replaying the recorded CHECK ADMIN message, waiting for a response that they have recorded. The corresponding CHECK RANDOM message

is then replayed, completing the challenge-response exchange. The attacker can then send one command which I have successfully used to add a new fingerprint or to reset the lock.

An attacker can try to get a repeated response from the lock 10 times per second, and on average will achieve 9 attempts per second (due to periodically having to re-connect to the lock) giving 32 400 attempts per hour. This attack can be performed with just one recorded exchange, however if an attacker records a series of exchanges they can shorten the expected time to perform this attack.

Figure 4.5 shows the probability of successfully executing this attack after a given period of time for varying numbers of recorded exchanges, computed as

$$1 - \left(\frac{65536 - n}{65536}\right)^{9t}$$

where n is the number of recorded exchanges and t is elapsed time in seconds. Figure 4.6 shows the amount of time required for test runs of this attack. Using a single recorded exchange is significantly slower than multiple exchanges, however having as few as 3 recorded exchanges provide relatively rapid successes. The attack can succeed rapidly without extended recording: the fastest observed success was 7 seconds, after requesting just 36 challenges.

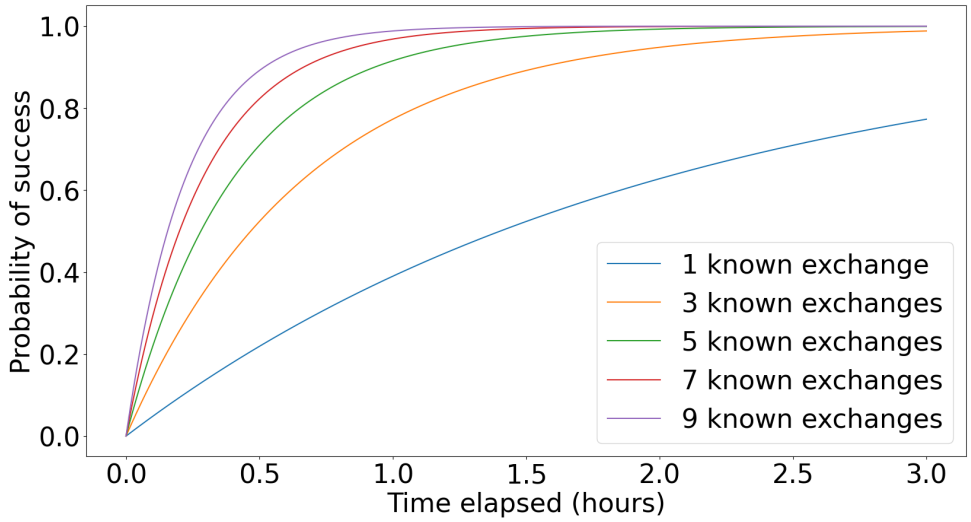


Figure 4.5: Probability of succeeding with the replay attack over time for different quantities of recorded challenge-response exchanges

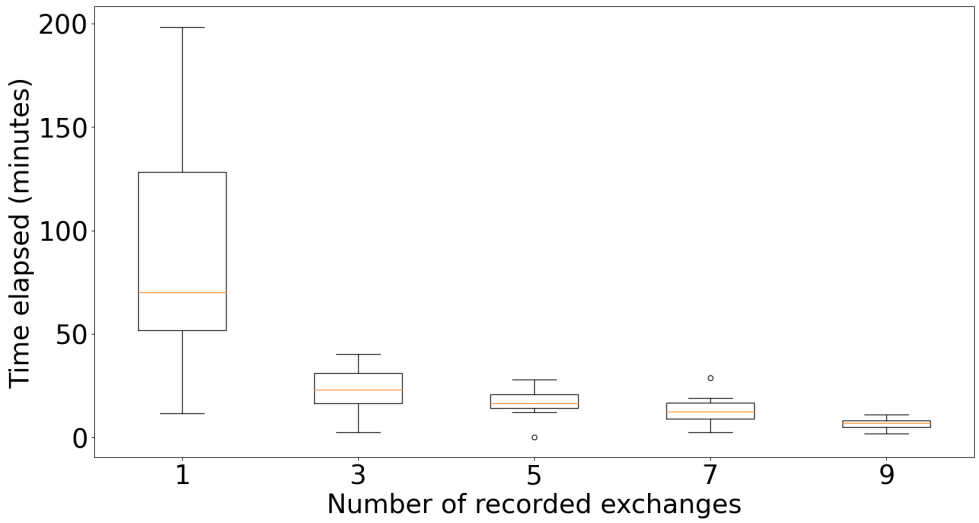


Figure 4.6: Times taken to run the replay attack for different quantities of challenge-response exchanges

Payloads

Many commands can be replayed after successfully performing this attack, provided they have been previously recorded. For example, the FP_MANAGE and IC_MANAGE messages for adding, modifying, or clearing stored fingerprints and RFID ("IC") cards can be replayed to repeat the operation, without the app knowing the new credentials. This allows an attacker to add a backdoor to the lock as credentials added in this manner do not show in the app until they are used to open the lock.

One important exception to this is the RESET command. RESET requires no data to be encrypted and sent with the message, and therefore an attacker can forge the RESET message without recording it using a MITM attack. A MITM is still required to perform the replay attack preceding the RESET message. Resetting the lock prevents the original user from being able to use the lock through the app, and a dedicated attacker can escalate this denial-of-service into a complete system lockout. The application provides a setting to disable the hard reset button (intended to prevent hotel guests from breaking the locks), and by disassembling the lock an attacker can unscrew the component which engages the backup lock, completely preventing usage by the owner and giving full control to the attacker.

Potential Solutions

This attack can be prevented by implementing proper replay protections against an attacker. The most appropriate method would be to add a timestamp to every

message: this mechanism is already in place for unlock messages and prevents this attack. Furthermore, the challenge-response protocol should be improved to avoid challenge re-use. A simple extension to the existing system is to make the challenges 4 bytes long instead of 2; the response already returns four bytes and the lock appears to use a secure random number generator, so this should be a simple change to make. This would multiple the expected number of challenges before a collision by a factor of 65536, making this attack infeasible without extended access to the lock.

4.4.3 Command Replacement Attack

The third discovered attack allows an attacker to replace a user command with their own. If an attacker is running an active BLE MITM attack, they can wait for a challenge-response exchange and then replace the user's command with a different payload, as discussed in Section 4.4.2. This is possible because the challenge-response exchange is separate from the command it authenticates, with a few exceptions.

One possible mitigation for this exploit would be to include the response to the lock's challenge in the command data, rather than sending a separate message. This is already used for the UNLOCK command, although not for any other command. Combined with the suggested replay preventions for the previous attack, an improved challenge-response protocol would be:

App → Lock: CHECK ADMIN (admin key, timestamp)
Lock → App: RESPONSE to CHECK ADMIN (success, timestamp, challenge)
App → Lock: COMMAND (sum(challenge, unlock key), timestamp, data)

4.4.4 Additional Design Weaknesses

There are several other weaknesses in the TTLock BLE protocol that have not lead to full exploits, however they are still potential security issues. I will briefly discuss each of them and their potential impact in turn.

Partial Encryption

The packets sent between the application and the lock contain many fields, however only the data section is encrypted. This means that the other information is public and can be modified by an attacker if desired. For example, the protocol version is sent unencrypted, which reveals out-of-date lock firmware, and could lead to downgrade attacks if modified by an attacker. Furthermore, the command sent is

exposed, allowing attackers to observe the operations being performed. Finally, the length of the encrypted data can be modified, which may be used to exploit buffer overflow vulnerabilities on some devices.

This can be fixed by encrypting the entire packet before transmission, instead of only encrypting one section. This will have a negligible impact on the performance of the system but provide increased security.

Absence of Message Authentication/Integrity Checking

The messages exchanged do not include a message authentication check. This means that an attacker can modify messages in transit, or forge their own messages, without any way for devices to know these messages are not legitimate. This can be improved by changing to an authenticated encryption mode such as GCM, or by adding a MAC to each message.

Chapter 5

Discussion

There are a variety of insights that can be made about smart lock security and lock design from the issues identified in this project. Both smart locks use outdated and vulnerable technologies as components of their system, despite the availability of more secure alternatives. Choosing technologies with different amounts of underlying security impacts the attack surface and the number of vulnerabilities found, illustrated by the difference in Wi-Fi and Bluetooth exploitation. There are several areas that appear to be difficult for developers to implement securely: exploits were found in the setup processes of both systems, and the BLE lock's attempt at using proper cryptography led to many different issues in the system. Furthermore, there is a lot of inconsistency in how different commands are secured, despite being part of the same protocol. Finally, developers appear to have difficulty implementing both the physical and cybersecurity elements properly, instead creating systems that are more secure in one element than the other.

5.1 Outdated Standards

System developers continue to use some outdated standards for components of their system which are known to have vulnerabilities. In this project, this primarily concerns the choice of both smart lock manufacturers to use Mifare Classic smart cards, despite the system being exploitable since 2008. The use of vulnerable standards is a major security issue as one simple choice can allow an attacker to bypass the rest of the security mechanisms implemented in the product.

From a security economics perspective, the use of outdated standards is easier to understand. The old version of the system has been around for longer, and therefore

benefits from network effects: there are many existing users, and a large amount of both required hardware for the systems and software for developers to integrate the technologies into their own systems. This makes the system far more likely to be chosen by developers, which in turn leads to a larger user base and the continued widespread use of the system. Furthermore, if a company is already using a certain standard, it is much easier for them to continue with that standard by re-using software they have written than to switch their product over to the new version.

Given the range of incentives to use vulnerable standards, there need to be economic drives for system designers to use more secure standards. For example, replacing the smart cards with transition standards such as Mifare Plus allows users and developers to continue using their existing software while implementing the more secure system, and once it is ready they can easily upgrade their software and transition the smart cards to the higher security level.

5.2 Underlying Security

It is much harder for attackers to find vulnerabilities in systems when the underlying technologies are secure. The higher level of protection in Wi-Fi — due to both link layer WPA encryption and transport layer security — lead to only a single Wi-Fi exploit in this project, which is only usable in a niche scenario and does not compromise the system. Technologies with less underlying security such as Bluetooth, which provides no end-to-end security and is vulnerable to man-in-the-middle attacks, are inherently easier to attack. This lead to a wide range of issues being discovered in the Bluetooth lock tested in this project. This implies that developers should attempt to rely on more secure technologies to improve the security of their system as a whole.

5.3 Setup

One common problem in both systems is the setup process. The Pineworld lock's Wi-Fi setup process leaks the network password to an attacker with an evil twin network, and the Ruveno lock's key exchange during the application's setup is trivial for a man-in-the-middle to attack. This first-time setup appears to consistently be a difficult process to perform securely, and therefore greater care should be given when designing the protocol.

There are various reasons for setup being a difficult problem. There are some

processes for which there is no existing standard (secure) method: for example, sharing credentials with a device that is not on the same network is difficult to achieve, but required for the setup of all Wi-Fi-based IoT devices.

5.4 Cryptography

Cryptography is a broad, complex subject that is difficult to learn in its entirety. It is unreasonable to expect all developers to have a full understanding of cryptography, and therefore it is widely recommended for developers to avoid creating their own cryptographic functions. For this reason, there are well-defined standards for various cryptographic primitives and their parameters, as well as dedicated cryptography libraries to ensure that developers do not introduce additional vulnerabilities in their own implementations.

Even when using standard algorithms and existing implementations, there are still issues that can arise from the misuse of cryptographic primitives. For example, many IoT developers will use AES so that they can claim "military-grade encryption", however developers must also select a block cipher mode and generate any required parameters such as initialisation vectors and nonces. Many block cipher modes have known vulnerabilities if improperly used, and other issues may arise from incorrect generation of IVs and nonces.

We see developers attempting to implement their own cryptographic solutions or shortcutting the standards which are known to be secure. In this project, this is most prominent with the TTLock BLE protocol: they have poorly re-invented key exchange, introduced vulnerabilities through improper use of AES, created an insecure challenge-response protocol which is exploited in two different attacks, and abandoned message authentication checks. While several of their misuses of cryptography can be explained as reducing the amount of data sent and the complexity of their code, the vulnerabilities that arise from these design choices are problematic and arguably not worth the benefits.

5.5 Consistency

In the TTLock BLE protocol, there is a lot of inconsistency in the security measures used for different commands. Notably, there appears to have been a larger focus on securing the UNLOCK function — it includes the response to CHECK ADMIN in the message data rather than using a separate message, preventing the command

replacement attack; it also includes the current timestamp in each message, preventing replay attacks. Despite the better security for the UNLOCK command, the same preventions are seen in very few other commands, making them vulnerable to the attacks presented in Section 4.4.

This disparity reflects the priorities of the developers: the most obvious attack on the lock is the possibility of someone unlocking it, so that command needs to be as secure as they can make it. They do not see the need for other commands to receive the same level of security even though they can also lead to the same compromise, which then leaves the commands vulnerable.

5.6 Balancing Physical and Cyber Security

Smart locks need to implement effective physical security as well as cyber security. Traditionally, companies tend to perform better with one area than the other: companies either understand physical security from their existing products and try to make their products "smart", or IoT developers want to add smart locks to their suite of existing smart devices. In the two systems analysed, there are both physical and cyber security issues with both systems, and in each case the vulnerabilities are mostly with one of these two areas: the Pineworld lock has several physical security issues and few technical exploits, while the Ruveno lock has very few physical issues but bountiful cyber security problems. This implies that the trend continues, even with "state-of-the-art" systems.

Chapter 6

Summary and Conclusions

In this project I have analysed a pair of smart locks, with one communicating over Wi-Fi and the other using Bluetooth. I have found attacks against multiple attack surfaces on both locks, including smart card cloning, weak backup locks, multiple fingerprint sensor attacks, an information leak based on Wi-Fi setup, and a wide range of protocol vulnerabilities in the TTLock BLE communications. Many of the technical vulnerabilities found are more complex, less powerful, and harder to exploit than the abundance of simple attacks found by previous smart lock security analyses; however, there are still some very powerful attacks on the smart locks, and several attacks which are trivial to exploit.

The majority of the technical exploits found are in the TTLock BLE communications; specifically, their use of cryptography. Standard key exchange algorithms have been replaced with their own method, which is trivial to attack. Their challenge-response protocol is also vulnerable to two separate attacks, which make use of a wide variety of design issues to create separate attacks. There are other design issues present that cannot be exploited by themselves, however they are indicative of possible vulnerabilities that I have not been able to exploit in this dissertation.

There are several areas that are hotspots for vulnerabilities. Initial setup procedures in both systems have been attacked, suggesting that this is a difficult process to perform securely. Both locks used insecure technologies as components of their system, allowing attackers to bypass the other more secure areas of the system. On the Bluetooth lock, there are a large number of issues in their cryptography which lead to a variety of attacks.

There are several broader issues exposed by this project. The technologies used for the smart locks provide varying levels of security, which prevent certain attack

surfaces being exposed to hackers. The lack of security provided by BLE allows attackers to break the insecure application-level protocol between devices, whereas the majority of traffic on the Wi-Fi lock uses TLS and cannot be attacked in a similar way. Consistency throughout an application is also an issue, with some commands on the BLE lock already implementing suggested defences against the attacks presented in Section 4.4 while others are vulnerable to them.

One continuing problem is the balance between physical security and cybersecurity. System designers tend to understand and implement one of the two areas far better than the other, and this trend continues with the two locks analysed. One system has three physical design flaws and two technical while the other only has two physical weaknesses and a wide range of technical vulnerabilities.

In summary, I have shown that even state-of-the-art smart lock systems are not perfectly secure. There are a wide range of problems that attackers can take advantage of for assorted malicious purposes, extending beyond simply unlocking the lock to gain access. Furthermore, there are several underlying issues that make the problem of implementing a secure system more difficult, ranging from the inherent security of the technologies in use to the difficulty of implementing certain subsystems securely. Although these systems are more secure than their predecessors, there is still work to be done in multiple aspects to make these systems more secure.

Bibliography

[1] A. Tierney, PenTestPartners, "Drilling open a smart door lock in 4 seconds." https://www.pentestpartners.com/security-blog/drilling-open-a-smart-door-lock-in-4-seconds/, 2019.

[2] A. Rose and B. Ramsey, "Picking Bluetooth Low Energy Locks from a Quarter Mile Away." Defcon 24, 2016.

[3] Jmaxxz, "Backdooring the Frontdoor." Defcon 24, 2016. Accessed on the 30th of April 2021.

[4] M. Ye, N. Jiang, H. Yang, and Q. Yan, "Security analysis of Internet-of-Things: A case study of August smart lock," in *2017 IEEE Conference on Computer Communications Workshops*, pp. 499–504, 2017.

[5] M. Fuller, M. Jenkins, and K. Tjølsen, "Security Analysis of the August Smart Lock." Massachusetts Institute of Technology, 2017.

[6] A. Tierney, PenTestPartners, "Totally pwning the Tapplock smart lock." https://www.pentestpartners.com/security-blog/totally-pwning-the-tapplock-smart-lock/, 2018.

[7] R. Felch, Black Hills Information Security, "Reverse engineering a smart lock." https://www.blackhillsinfosec.com/reverse-engineering-a-smart-lock/, 2020.

[8] K. Nohl, D. Evans, S. Starbug, and H. Plötz, "Reverse-Engineering a Cryptographic RFID Tag.," in *USENIX security symposium*, vol. 28, 2008.

[9] F. D. Garcia, G. de Koning Gans, R. Muijrers, P. van Rossum, R. Verdult, R. W. Schreur, and B. Jacobs, "Dismantling MIFARE Classic," in *Computer Security - ESORICS 2008*, pp. 97–114, Springer Berlin Heidelberg, 2008.

[10] G. de Koning Gans, J.-H. Hoepman, and F. D. Garcia, "A Practical Attack on the MIFARE Classic," in *Smart Card Research and Advanced Applications*, pp. 267–282, Springer Berlin Heidelberg, 2008.

[11] F. D. Garcia, P. van Rossum, R. Verdult, and R. W. Schreur, "Wirelessly Pickpocketing a Mifare Classic Card," in *2009 30th IEEE Symposium on Security and Privacy*, pp. 3–15, 2009.

[12] C. Meijer and R. Verdult, "Ciphertext-only cryptanalysis on hardened Mifare classic cards," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pp. 18–30, 2015.

[13] D. Oswald and C. Paar, "Breaking Mifare DESFire MF3ICD40: Power analysis and templates in the real world," in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 207–222, Springer, 2011.

[14] D. Hurley-Smith and J. Hernandez-Castro, "Bias in the mifare DESFire EV1 TRNG," in *International Workshop on Radio Frequency Identification: Security and Privacy Issues*, pp. 123–133, Springer, 2016.

[15] R. Flynn, "An investigation of possible attacks on the MIFARE DESFire EV1 smartcard used in public transportation," 2019.

[16] J. Padgette, J. Bahr, M. Batra, M. Holtmann, R. Smithbey, L. Chen, and K. Scarfone, "Guide to Bluetooth Security," 2017-05-08 2017.

[17] M. Jakobsson and S. Wetzel, "Security Weaknesses in Bluetooth," pp. 176–191, Springer-Verlag, 2001.

[18] F.-L. Wong, F. Stajano, and J. Clulow, "Repairing the Bluetooth Pairing Protocol," in *Security Protocols* (B. Christianson, B. Crispo, J. A. Malcolm, and M. Roe, eds.), (Berlin, Heidelberg), pp. 31–45, Springer Berlin Heidelberg, 2007.

[19] D. Antonioli, N. O. Tippenhauer, and K. B. Rasmussen, "The KNOB is Broken: Exploiting Low Entropy in the Encryption Key Negotiation Of Bluetooth BR/EDR," in *28th USENIX Security Symposium (USENIX Security 19)*, (Santa Clara, CA), pp. 1047–1061, USENIX Association, Aug. 2019.

[20] A. Stubblefield, J. Ioannidis, and A. Rubin, "Using the Fluhrer, Mantin, and Shamir Attack to Break WEP," in *NDSS*, 2002.

[21] E. Tews, "Attacks on the WEP protocol," 2007.

[22] E. Tews and M. Beck, "Practical attacks against WEP and WPA," in *Proceedings of the second ACM conference on Wireless network security*, pp. 79–86, 2009.

[23] M. Kraft and J. Holston, "A Targeted Wireless Denial of Service Attack: Deauth or not to Deauth, That is the Question," in *Proceedings of the 7th International Conference on Information Warfare and Security: ICIW*, p. 148, Academic Conferences Limited, 2012.

[24] M. Vanhoef and F. Piessens, "Key Reinstallation Attacks: Forcing Nonce Reuse in WPA2," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, (New York, NY, USA), p. 1313–1328, Association for Computing Machinery, 2017.

[25] M. Vanhoef and E. Ronen, "Dragonblood: Analyzing the Dragonfly handshake of WPA3 and EAP-pwd," in *IEEE Symposium on Security & Privacy (SP)*, IEEE, 2020.

[26] "Computer Misuse Act 1990, c. 18. Available at: https://www.legislation.gov.uk/ukpga/1990/18/contents (Accessed: 3 May 2021)."

[27] "Copyright, Designs and Patents Act 1988, c. 48. Available at: https://www.legislation.gov.uk/ukpga/1988/48/contents (Accessed: 3 May 2021)."

[28] "The Copyright and Related Rights Regulations 2003. Available at: https://www.legislation.gov.uk/uksi/2003/2498/contents (Accessed: 3 May 2021)."

[29] P. Rascagneres and V. Ventura, "Fingerprint cloning: Myth or reality?." https://blog.talosintelligence.com/2020/04/fingerprint-research.html, 2020.