

Helene Sjøgren Bolkan
Jeanette Hue Nhu Tran

Security Analysis of the Mobile Applications Used in the Pacemaker Ecosystem

Master's thesis in Communication Technology

Supervisor: Marie Moe

Co-supervisor: Guillaume Bour

June 2023



Norwegian University of
Science and Technology

Helene Sjøgren Bolkan
Jeanette Hue Nhu Tran

Security Analysis of the Mobile Applications Used in the Pacemaker Ecosystem

Master's thesis in Communication Technology
Supervisor: Marie Moe
Co-supervisor: Guillaume Bour
June 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology



Title: Security Analysis of the Mobile Applications Used in the Pacemaker Ecosystem

Students: Bolkan, Helene Sjøgren and Tran, Jeanette Hue Nhu

Problem description:

The domain of healthcare is constantly evolving, and the use of connected technology is increasingly common. Within the field of Implantable Medical Devices (IMDs), the use of mobile applications is an expansion of the ecosystem. This is observed in several areas, also within this thesis' focus field, namely the pacemaker ecosystem. We aim to look into the new mobile applications connected to pacemakers. In particular, we will focus on the data privacy and data security issues these might bring.

Several vendors of pacemakers have incorporated mobile applications in various forms, both as a substitute for a dedicated device for remote monitoring and as an added benefit for the patient. The Home Monitoring Unit (HMU) is a device responsible for gathering data from the pacemaker and forwarding it to the vendor's servers. The vendors who choose the substitute approach will now have a mobile application to replace the HMU. The attack surface changes when this gateway is switched to a mobile application. For other vendors, the mobile application is just an addition to the service for the patient. Here, the mobile application does not communicate directly with the pacemaker. In both cases, there are potentially new security vulnerabilities introduced to the system.

To assess the security of the system, threat modeling will be performed. Additionally, we will conduct an analysis of the vendors' mobile applications. The focus throughout the thesis will be on the patient's safety and privacy. We will compare the security of the older system containing the hardware HMU with the new system containing the mobile application. Additionally, to include more insight from a patient's point of view, a questionnaire study will be carried out. The combination of patient insights and technical security analysis will give us valuable insight into the potential security challenges of the new version of the pacemaker ecosystem.

Approved on: 2023-03-22

Main supervisor: Associate Professor Moe, Marie, NTNU

Co-supervisor: Bour, Guillaume, SINTEF

Abstract

Our increasing connectivity on the Internet has led to a technology dependence, which is growing faster than our ability to secure it. This poses significant risks for people using these devices. The pacemaker industry connects its life-critical devices to the Internet to improve the quality of life, which exposes new attack surfaces. Researchers have demonstrated vulnerabilities and weak security measures in pacemakers, and companies producing these devices have been exposed for not having good security practices. The recent innovation within this field introduces mobile applications connected to the pacemaker to provide detailed insight to the patients. The app offers benefits in terms of patient monitoring and data accessibility. However, it also introduces new attack vectors that can compromise the security and privacy of the patient. In this thesis, we investigate the cyber security concerns connected to mobile applications used in the pacemaker ecosystem focusing on patient safety and privacy. We conduct a threat modeling of the ecosystem provided by two of the main manufacturers to identify and analyze different threats that can exist in the new ecosystem. Further, we perform a security analysis on mobile applications to find potential vulnerabilities that can be exploited by an attacker. The results describe new attack surfaces that affect the patient's privacy, such as outdated signature algorithms, weak password policy, and the possibility of impersonating a mobile application. These are compared with the old pacemaker ecosystem. This includes a hardware device as the dedicated communication gateway. Another research finding is gathering user insight by conducting a questionnaire study sent out to patients with a pacemaker. It suggests that the respondents were positive about the mobile applications, but that their habits and awareness may decrease the security level. Our findings contribute to the understanding of mobile apps connected to a medical device, offering new and valuable insights in this area. In addition, we highlight the importance of conducting comprehensive security testing of these mobile apps and analyzing potential consequences before deploying them into the world, as these mobile apps handle personal and sensitive data. Security testing is therefore not just a recommended practice, but a critical necessity.

Sammendrag

Vår økende tilkobling på internett har ført til en teknologiavhengighet, som øker raskere enn vi er i stand til å sikre den. Dette medfører betydelige risikoer for personer som bruker disse enhetene. Pacemaker-industrien kobler sine livskritiske enheter til internett for å forbedre livskvaliteten, noe som resulterer i nye angrepsflater. Forskere har klart å finne sårbarheter og svake sikkerhetstiltak i pacemakere, og selskaper som produserer disse enhetene har blitt avslørt for å ikke ha gode sikkerhetsrutiner. Den nyeste innovasjonen innenfor dette feltet introduserer mobilapplikasjoner som er koblet til pacemakeren, for å gi informasjon til pasienten. Appen gir fordeler i form av pasientovervåking og tilgjengelighet til data. Imidlertid introduserer den også nye angrepsvektorer som kan true sikkerheten og personvernet til pasienten. I denne oppgaven undersøker vi mobilapplikasjoner som brukes i pacemaker-økosystemet, med fokus på sikkerheten til pasienten. Vi gjennomfører en trusselmodellering av økosystemet til hver mobilapplikasjon for å identifisere og analysere forskjellige trusler som potensielt kan eksistere i det nye økosystemet. Vi utfører deretter en sikkerhetsanalyse av mobilapplikasjonene for å finne potensielle sårbarheter som kan utnyttes av en angriper. Resultatene beskriver nye angrepsflater som påvirker pasientens personvern, for eksempel utdaterte signaturalgoritmer, svake regler for passord og muligheten for å utgi seg for å være en mobilapplikasjon. Disse resultatene sammenlignes med det gamle pacemaker-økosystemet, som inkluderer en maskinvareenhet som den dedikerte kommunikasjonsporten. Vi har i tillegg utført innsamling av brukerinnsikt. Dette er gjennomført med en spørreundersøkelse som er sendt ut til pasienter med pacemaker. Det viser seg at respondene var positive til mobil applikasjonene, men at deres vaner og bevissthet kan redusere sikkerhetsnivået. Våre funn bidrar til økt forståelse av mobilapplikasjoner som er koblet til medisinsk utstyr og det gis ny og verdifull innsikt på dette området. I tillegg understreker vi viktigheten av å gjennomføre omfattende sikkerhetstesting av disse mobilappene og analysere potensielle konsekvenser før de tas i bruk, ettersom disse mobilappene håndterer personlig og sensitiv informasjon. Sikkerhetstesting er derfor ikke bare en anbefalt praksis, men en kritisk nødvendighet.

Preface

This Masters's Thesis is the final deliverable of Helene Bolkan and Jeanette Tran of their Master of Science degrees in Communication Technology, with a specialization in Information Security. The thesis is written at the Department of Information Security and Communication Technology, Norwegian University of Science and Technology (NTNU). The pre-project work is performed in the Autumn of 2022, while the thesis is written in the Spring of 2023.

This thesis is a collaboration between NTNU and SINTEF, a research organization in Trondheim. It is part of a series of master theses about the security of medical devices, all led by Associate Professor Marie Moe.

Acknowledgments

First, we would like to give a big thanks to NTNU and SINTEF for offering us equipment. Extra thanks to SINTEF for proposing us an office and a good coffee machine with milk 80% of the time.

A big appreciation to our supervisor and responsible professor, Marie Elisabeth Gaup Moe, for proposing such an exciting topic. It has been an educational journey and we are grateful that you introduced us to the security of medical devices and made us knowledgeable in this field. We have enjoyed working with something that has such an impact on people and must thank you for this opportunity.

Much appreciation to our co-supervisor Guillaume Nicholas Bour, for guiding us through the thesis and always being available for questions and motivational words when we were frustrated, especially at the beginning of the process of pen-testing. Thank you for teaching us different methods and techniques for security testing and how to report findings. We greatly appreciate this knowledge as it will be highly valuable in our future cyber security jobs.

Lastly, we are extremely thankful for our classmates. Every day we went through this master period together with daily quizzes and good coffee during lunch break. We truly appreciate your presence as it improved our well-being during this period. Writing a master's thesis would not have been as fun without you.

Contents

| | |
|--|-------------|
| List of Figures | xiii |
| List of Tables | xv |
| List of Acronyms | xvii |
| 1 Introduction | 1 |
| 1.1 Context | 1 |
| 1.1.1 Implantable Medical Devices | 1 |
| 1.1.2 The Pacemaker Ecosystem | 2 |
| 1.2 Motivation | 4 |
| 1.3 Scope of the Project | 5 |
| 1.4 Research Questions | 6 |
| 1.5 Structure of the Thesis | 6 |
| 2 Related Work | 9 |
| 2.1 Previous Work on Mobile Application Security | 9 |
| 2.2 Previous Work on Pacemaker Hacking | 11 |
| 3 Technical Background | 13 |
| 3.1 Relevant Guidelines | 13 |
| 3.2 Standards | 14 |
| 3.3 Security Terminology | 15 |
| 3.4 Android Applications | 18 |
| 4 Methodology | 21 |
| 4.1 Threat Modeling | 21 |
| 4.1.1 Datagram Flow Diagrams | 22 |
| 4.1.2 STRIDE | 22 |
| 4.1.3 Mitigation Techniques | 24 |
| 4.2 Black Box Testing | 26 |
| 4.3 Analysis of Applications | 27 |
| 4.3.1 Extracting APK Files | 27 |

| | | |
|----------|--|------------|
| 4.3.2 | Decompile APK File to JAR File | 28 |
| 4.3.3 | Tools for Security Analysis | 28 |
| 4.4 | Sending out Questionnaires | 31 |
| 4.4.1 | Finding Questions | 32 |
| 4.5 | Ethical Concerns | 33 |
| 5 | Threat Modeling | 35 |
| 5.1 | Medtronic’s Ecosystem | 36 |
| 5.1.1 | What Are We Working On? | 36 |
| 5.1.2 | What Can Go Wrong? | 37 |
| 5.1.3 | What Are We Going to Do About It? | 46 |
| 5.2 | Biotronik’s Ecosystem | 52 |
| 5.2.1 | What Are We Working On? | 52 |
| 5.2.2 | What Can Go Wrong? | 52 |
| 5.2.3 | What Are We Going to Do About It? | 57 |
| 6 | Security Analysis of Applications | 59 |
| 6.1 | MyCareLink Heart App - Medtronic | 60 |
| 6.1.1 | Static Analysis | 60 |
| 6.1.2 | Permissions | 70 |
| 6.1.3 | Analysis of the APK | 71 |
| 6.1.4 | Structure of the Mobile Application | 75 |
| 6.1.5 | Summary of our Findings | 77 |
| 6.2 | Patient App - Biotronik | 78 |
| 6.2.1 | Static Analysis | 79 |
| 6.2.2 | Permissions | 83 |
| 6.2.3 | Structure of the Mobile Application | 85 |
| 6.2.4 | Client Bypass Authentication | 93 |
| 6.2.5 | Application Authentication | 94 |
| 6.2.6 | Certificate Details | 98 |
| 6.2.7 | Unverified Issues | 99 |
| 6.2.8 | Summary of our Findings | 101 |
| 7 | Questionnaire | 103 |
| 7.1 | Demographics and Clinical | 104 |
| 7.2 | Mobile Phone Usage | 105 |
| 7.3 | Perception of Applications Connected to IMDs and Cybersecurity | 108 |
| 8 | Discussion | 115 |
| 8.1 | Security of the Mobile Application System | 115 |
| 8.1.1 | Discussion on Findings from Medtronic’s System | 115 |
| 8.1.2 | Discussion on Findings from Biotronik’s System | 120 |

| | | |
|----------|--|------------|
| 8.2 | Benefits and Drawbacks of the Mobile Applications | 125 |
| 8.2.1 | Positive Aspects of Changing to an Application | 125 |
| 8.2.2 | Negative Aspects of Changing to an Application | 126 |
| 8.3 | Hardware HMU vs. Mobile Applications | 127 |
| 8.3.1 | Security of the HMU System | 127 |
| 8.3.2 | Comparison of the Systems | 129 |
| 8.4 | Actual Security vs. Perceived Security From Patients | 133 |
| 8.5 | Limitations of our Work | 136 |
| 8.6 | Future work | 137 |
| 9 | Conclusion | 139 |
| | References | 141 |
| A | Tools and Procedures | 149 |
| A.1 | Decompiling an APK File to a JAR File | 149 |
| A.2 | Create and Run an Emulator | 149 |
| A.3 | How to Install ADB | 150 |
| A.4 | Configure the Emulator to Work with Burp Suite | 150 |
| A.5 | Configure the Emulator to Work with Mitmproxy | 151 |
| A.6 | Intercept a Request with Burp Suite | 152 |
| A.7 | Intercept a Request with Mitmproxy | 152 |
| A.7.1 | The Script and Corresponding JSON Files | 153 |
| A.8 | Setting Up Frida | 155 |
| B | Questionnaire | 157 |
| B.1 | The Questions | 157 |

List of Figures

| | | |
|------|--|-----|
| 1.1 | The pacemaker ecosystem | 3 |
| 5.1 | Data flow diagram of Medtronic’s ecosystem | 37 |
| 5.2 | Impact matrix | 39 |
| 5.3 | Data flow diagram of the Biotronik’s ecosystem | 53 |
| 6.1 | The Google API key and Google Crash Reporting API key in strings.xml | 69 |
| 6.2 | Fatal Exception error when using Frida | 74 |
| 6.3 | Certificate details of Medtronic | 76 |
| 6.4 | TLS-traffic in Wireshark for PatientApp, the Biotronik mobile application | 86 |
| 6.5 | Authentication request in Burp Suite | 87 |
| 6.6 | Logcat in Android Studio | 87 |
| 6.7 | The mitmweb interface | 89 |
| 6.8 | Logcat in Android Studio for PatientApp | 90 |
| 6.9 | One of the server’s websites displaying KeyCloak | 91 |
| 6.10 | IP WHOIS Lookup of one of the domain names of Biotronik | 92 |
| 6.11 | A sequence diagram showing a device trying to log in to the application, with communication with the servers in Biotronik | 93 |
| 6.12 | The final process of registering a user | 95 |
| 6.13 | The requests in mitmproxy with 200 OK status responses | 95 |
| 6.14 | The ClientSecret and ClientId in strings.xml | 96 |
| 6.15 | Header of the request going to the authorization server | 97 |
| 6.16 | Decoding of a string with Base64 encoding | 97 |
| 6.17 | Certificate details of Biotronik | 99 |
| 6.18 | The result from running Frida with the Biotronik app. | 101 |
| 7.1 | Graphs of Demographics | 105 |
| 7.2 | The result of the Likert scale questions on mobile phone usage | 107 |
| 7.3 | Graph of how the respondents make their passwords | 108 |
| 7.4 | The result of the Likert scale questions on perception of application connected to IMDs and cybersecurity | 109 |
| 7.5 | The respondents’ thoughts of an application connected to their pacemaker | 110 |

| | | |
|-----|---|-----|
| 7.6 | The respondents answer on what would make them consider an application connected to a pacemaker | 112 |
| A.1 | Emulator showing the option for Extended Controls | 151 |
| A.2 | Toggle the intercept button | 152 |
| A.3 | How to respond to a request | 153 |

List of Tables

| | | |
|------|---|----|
| 4.1 | Data Flow Diagram (DFD) objects | 23 |
| 4.2 | Description of the Spoofing, Tampering, Repudiation, Information disclosure, Denial of service and Elevation of privilege (STRIDE) elements with the associated security properties | 24 |
| 4.3 | Available applications from different manufacturers | 28 |
| 5.1 | Identified pacemaker components and their description | 38 |
| 5.2 | Description of STRIDE threats against the pacemaker device and their impact | 39 |
| 5.3 | Description of STRIDE threats against the pacemaker app and the mobile phone and their impact | 41 |
| 5.4 | Description of STRIDE threats against the data server and their impact | 42 |
| 5.5 | Description of STRIDE threats against the doctor’s device and their impact | 43 |
| 5.6 | Description of STRIDE threats against the communication over the Internet and their impact | 44 |
| 5.7 | Description of STRIDE threats against the communication between the pacemaker and phone and their impact in the Medtronic ecosystem | 45 |
| 5.8 | Summary of STRIDE categories with identified threats against the pacemaker’s components in the Medtronic ecosystem | 45 |
| 5.9 | Identified pacemaker components and their description in the Biotronik’s ecosystem | 54 |
| 5.10 | Description of STRIDE threats against the hardware HMU and their impact in Biotronik | 56 |
| 5.11 | Description of STRIDE threats against the communication between HMU and pacemaker and their impact in Biotronik | 56 |
| 5.12 | Summary of STRIDE categories with identified threats against the pacemaker’s components in the Biotronik ecosystem | 57 |
| 6.1 | The result of the code analysis by Mobile Security Framework (MobSF) of MyCareLink Heart App and Biotronik | 65 |
| 6.2 | The result of the code analysis by BeVigil of MyCareLink Heart App | 67 |
| 6.3 | The summary of findings for MyCareLink Heart App | 78 |
| 6.4 | The result of the code analysis by BeVigil of Patient App | 81 |

| | | |
|-----|---|-----|
| 6.5 | The summary of findings for PatientApp | 102 |
| 8.1 | Comparison between the HMU and the mobile application from Medtronic | 130 |
| 8.2 | Comparison between the HMU and the combination of HMU and mobile application from Biotronik | 132 |

List of Acronyms

- ADB** Android Debug Bridge.
- AES** Advanced Encryption Standard.
- APK** Android Package Kit.
- AVD** Android Virtual Device.
- BLE** Bluetooth Low Energy.
- CIA** Confidentiality, Integrity and Availability.
- CRT** Cardiac Resynchronization Therapy.
- CWE** Common Weakness Enumeration.
- DES** Data Encryption Standard.
- DEX** Dalvik EXecutable.
- DFD** Data Flow Diagram.
- DNS** Domain Name System.
- DoS** Denial of Service.
- EoP** Elevation of Privilege.
- FIPS** Federal Information Processing Standards.
- FISMA** Federal Information Security Management Act.
- FLE** Field-Level Encryption.
- GUI** Graphical User Interface.
- HMU** Home Monitoring Unit.

HTTP Hypertext Transfer Protocol.

HTTPS Hypertext Transfer Protocol Secure.

IAM Identity Access Management.

ICD Implantable Cardioverter-Defibrillator.

IMD Implantable Medical Device.

IoMT Internet of Medical Things.

IoT Internet of Things.

JAR Java ARchive.

JWT JSON Web Token.

MASTG Mobile Application Security Testing Guide.

MASVS Mobile Application Security Verification Standard.

MICS Medical Implant Communication System.

MITM Man-in-the-middle.

MobSF Mobile Security Framework.

NIST National Institute of Standards and Technology.

NTNU Norwegian University of Science and Technology.

OIDC OpenID Connect.

OS Operating System.

OWASP Open Web Application Security Project.

RNG Random Number Generator.

SSL Secure Sockets Layer.

SSO Single Sign-On.

STRIDE Spoofing, Tampering, Repudiation, Information disclosure, Denial of service and Elevation of privilege.

TLS Transport Layer Security.

VPMN Virtual Private Mobile Network.

VPN Virtual Private Network.

WHO World Health Organization.

Chapter 1

Introduction

1.1 Context

To provide the necessary background knowledge for our thesis, we start by presenting the context. This gives a short introduction to the field our thesis resides within, as well as an introduction to important concepts that are used later on.

1.1.1 Implantable Medical Devices

In order to obtain an understanding of the term IMD, we start by defining what a medical device implies. A concise definition of a medical device is the one given by World Health Organization (WHO): “A medical device can be any instrument, apparatus, implement, machine, appliance, implant, reagent for in vitro use, software, material or another similar or related article, intended by the manufacturer to be used, alone or in combination for a medical purpose. Such health technologies are used to diagnose illness, monitor treatments, assist disabled people, and intervene and treat illnesses, both acute and chronic”. According to WHO there are an estimated 2 million different kinds of medical devices on the market worldwide [49]. The definition of a medical device is quite broad and can be summarized as any *thing* that is used for a medical purpose, to help a patient.

A category within the umbrella of medical devices is IMD. According to the Norwegian Medicines Agency, this is a device that is inserted either totally or partially into the human body. The device can be implanted medically or surgically and is intended to remain in the body after the procedure [41]. This means that an IMD is a device that is permanently placed within the human body to help with different medical conditions. These devices are quite intrusive, as they become a part of the human body, and are thus often used to help with conditions and illnesses that greatly impact the patient’s quality of life. Examples of medical conditions that an IMD can help with are diabetes, heart arrhythmia, and sleep apnea.

The main focus for this thesis is IMDs which is used for heart rate monitoring. For this purpose, we have two devices, a pacemaker and an Implantable Cardioverter-Defibrillator (ICD). These have many similarities, their main task is to keep the heart rhythm of the patient consistent, which is done by giving small electrical stimulations to the heart. The ICD also has the option to give larger shocks of electricity, in the case of life-threatening arrhythmia. As these devices are so similar from a security perspective, we refer to them both as *pacemakers* in this thesis. A pacemaker is a tool for patients that have different heart anomalies and helps with irregular heart rhythms.

The pacemaker is inserted into the patient's chest during surgery. The device itself is only a few centimeters wide and weighs around 100 grams. It is equipped with several wires that are attached to the heart, through which the electrical pulses pass. In addition, the pacemaker has a battery and a tiny computer circuit [63]. The pacemaker is able to communicate wirelessly with other devices in the pacemaker ecosystem so that data can be sent to the responsible doctor, or changes can be made without invasive surgery.

1.1.2 The Pacemaker Ecosystem

The pacemakers are not stand-alone devices, and there exists an entire ecosystem to support these IMDs. This ecosystem exists in order to make the pacemaker function efficiently as a wireless device. There are several devices that are of importance to the pacemaker ecosystem, and these are described below:

The Pacemaker is the IMD itself. This is the device inside the patient's body and the most important part of the ecosystem.

The Programmer is the device that is responsible for the configuration of the pacemaker. Whenever a pacemaker is inserted for the first time, or there is a need for a change in the configuration of the pacemaker, the programmer is used. This communicates wirelessly with the IMD, without the need for invasive surgery. The programmer is equipped with a programming head, as it has to be in close proximity to the patient when making changes to the software configuration.

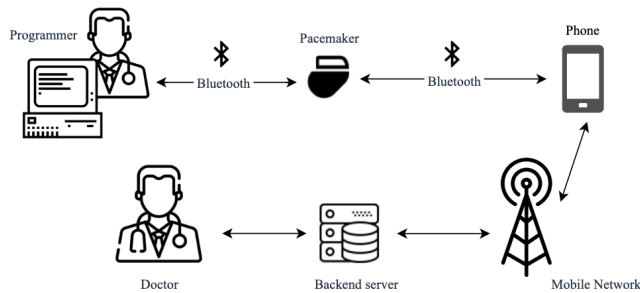
The Home Monitoring Unit is the gateway that receives data from the pacemaker before it is sent further to the patient's doctor. This device simplifies the patient's life, as the data can be accessed remotely by the doctor, which decreases the need for visits to the doctor's office. This gateway can take two different forms with the newest generation of pacemakers:

1. The gateway is an application in the patient's phone. This means that the patient can bring the gateway with her, and can give easier access to health data.
2. The gateway is dedicated hardware HMU, situated in the patient's home. This device is more stationary, and the patient will not necessarily have any insight into their heart data.

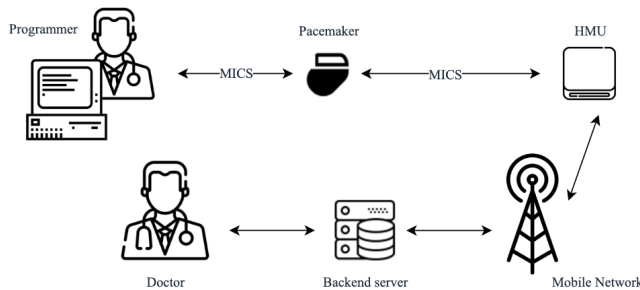
These two versions of the system can be seen in Figure 1.1.

The Mobile Network is part of the ecosystem, as it is the HMU's way of sending the data to the patient's doctor. Usually, the HMU uses the mobile network to send the data from the pacemaker to the backend servers of the doctor's office, which the doctor can access to get insight.

The Backend Servers are responsible for collecting and storing the patient data sent from the HMU. The data is sent over the mobile network. Usually, these servers can be accessed by doctors via an online platform.



(a) The new version of the pacemaker ecosystem with a phone



(b) The old version of the pacemaker ecosystem with an HMU

Figure 1.1: The pacemaker ecosystem

1.2 Motivation

The evolution of medical devices has been considered a fundamental component of the healthcare system. The primary medical device of our thesis, the pacemaker, saves lives by monitoring and coordinating heartbeat and increases the patient's lifetime. The drive to investigate these medical devices grows when they affect people to give them the possibility of healthier and longer lives.

As discussed in the pre-project, the safety of the patients is one important motivation factor for this thesis. A concern is that the pacemaker can be remotely controlled by unauthorized attackers, so it needs to be properly secured to maintain vital security assets. St. Jude Medical is an example of a company producing pacemakers that had bad practices regarding security. The company was exposed for having severe vulnerabilities in its pacemaker products. The researchers revealed that it was possible to do battery-draining and other tampering attacks on pacemakers by this particular vendor [45]. Consequently, doing research work on the cybersecurity part of the pacemaker is beneficial for society, which makes it highly motivating to contribute to this field by searching for other potential attacks and securing the medical device against them.

An article by VentureBeat, a website that provides tech news, announced that Medtronic launched an application that connects to the patient's phone to obtain data from their pacemakers, which was published in 2019 [53]. Due to the change in the pacemaker system recently happening in the last few years, the subject is quite untouched. In other words, there has not been done much research work on the healthcare applications connected to pacemakers. Hence, we are part of the early stage of this research field which is motivating.

Lastly, Internet of Things (IoT) is developing and will have a greater influence in the future. In the healthcare sector, we can see more and more medical devices switching over to operating wirelessly. The pacemaker is an example of a wirelessly medical device and is expected to increase in numbers in the next years. As discovered in the pre-project assigned in autumn 2022, it was predicted an increase of over 25% in the global use of the embedded device from the year 2016 to 2023, with 1.14 million units vs. 1.43 million units.¹ Similarly, the development of pacemakers has also had an impact in Norway with thousands of users. Statistics demonstrate that Norway's number of patients with implantation or replacement of pacemakers increased by 35% from 2012 to 2021, respectively 3519 units and 4788 units.² To summarize, the global use and the growth of the device are additional drives for the thesis - the larger the group of people it helps, the bigger the motivation.

¹<https://www.statista.com/statistics/800794/pacemakers-market-volume-in-units-worldwide/>

²<https://www.statista.com/statistics/978925/number-of-patients-with-implantation-or-replacement-of-pacemaker-in-norway/>

1.3 Scope of the Project

During our initial research, we noticed two distinct ways of incorporating a mobile phone into the pacemaker ecosystem. Either as an additional device to the HMU or as a replacement for the HMU. Therefore, we chose two different vendors for our investigation, namely Medtronic and Biotronik. The application from Medtronic acts as a substitute for the HMU. For Biotronik, the application is an additional device besides the HMU. In addition, previous master theses on this aspect have already discovered vulnerabilities on the HMU by Biotronik [12]. It has been proven that the HMU can be used to threaten a patient's safety and privacy. These results are used to compare with the new versions of the system, still with a focus on patient safety and privacy. They tested older versions of the system with the HMU, where our focus is the system containing a mobile phone.

Our project explicitly investigates the security of the mobile application used in the pacemaker ecosystem, which is a small part of the system. More specifically, we analyze the mobile applications from the vendors Medtronic and Biotronik. The two main surfaces outlined are:

Scope 1: The mobile application from Medtronic, which is a substitute for the HMU

Scope 2: The mobile application from Biotronik, which works in parallel with the HMU

It is important to note any further analysis conducted outside the mobile application, including the backend server and the communication link, falls outside the scope. However, they are mentioned as potential areas for analysis and are briefly discussed as part of our results. The other components of the pacemaker ecosystem, such as the programmer and the pacemaker itself, are completely out of scope, and will not be analyzed nor mentioned.

In addition to changing from a hardware HMU to a mobile application, there is a change in the communication protocol used by the vendors. Given the complexity of transitioning from the old proprietary protocol to a new Bluetooth protocol, focusing on both the mobile applications and the switch of the protocol is not possible. Therefore, we choose only to do mobile application security testing, more specifically Android application testing. The goal is to analyze mobile applications and find potential security vulnerabilities that can affect the patient's life.

1.4 Research Questions

One research question is defined for our contribution to the research field concerning the security of the pacemaker ecosystem.

What are the main cyber security concerns for mobile applications in the pacemaker ecosystem when it comes to patient safety and privacy?

With this research question as the foundation, we can derive these research objectives:

O1: Analyzing the mobile healthcare applications to see if they have an adequate level of security regarding patient data and privacy.

O2: Identifying and analyzing different threats to this new version of the pacemaker ecosystem.

O3: Gathering information about the patient's perspective by executing a survey and analyzing the results.

These objectives are elaborated in further detail in the upcoming chapters.

1.5 Structure of the Thesis

The remaining chapters are structured in the following manner:

Chapter 2 presents related work to provide the reader with necessary and related studies that have been performed on medical devices. Both previous works on mobile application security and pacemaker hacking are supplied to give the foundation of our research.

Chapter 3 provides the reader with a technical background that is useful to understand the thesis. In this chapter, multiple security concepts, guidelines, standards, and terminologies are presented.

Chapter 4 gives a further understanding of our work by presenting the methodologies and how they are used in general. In addition, we provide motivations and arguments for why we chose these particular methodologies.

Chapter 5 is the start of the analysis, where we perform threat modeling. In this chapter, we perform a threat modeling containing the system models for two vendors, the vulnerabilities and potential attacks, and the applicable mitigation techniques. There is one threat modeling for the Medtronic mobile application and another one for the Biotronik mobile application.

Chapter 6 contains the security analysis of mobile applications. Here, the two different apps are divided into their own sections. Different analyses are performed in this chapter, utilizing several tools, and the results are presented.

Chapter 7 sheds light on the patients, as we perform a questionnaire and present the results in this chapter. This gives insight into the patient's perspective on these mobile applications.

Chapter 8 provides a discussion on the results from chapters 6 and 7. The research question is answered in this chapter. We look into the implications of our work, and the perceived security from patients, and make a comparison of the older version of the ecosystem with the newer mobile application version.

Chapter 9 gives a short conclusion of our work in total. Here we summarize our findings and the implications of our work.

To enhance clarity and readability, appendix A provides additional information about the tools utilized during the security analysis and a detailed explanation of the procedure. The second appendix B provides the complete list of questions and the alternatives used in chapter 7.

Chapter

Related Work

Within the fields of healthcare technology and application security, there have been several contributions before our thesis. The following sections address some of the previous work that is done within the pacemaker ecosystem and validations of its security, and some previous work on mobile application security analysis. We present in more detail some of the work already covered in our project work [84].

2.1 Previous Work on Mobile Application Security

As mentioned, the new addition to the pacemaker ecosystem is the ability to use a mobile application as the gateway between the pacemaker and the doctor. When it comes to mobile applications, there are several research angles one could focus on. In order to limit the scope of this thesis, we decided to focus on either applications targeted towards Android phones or Apple phones. These two Operating Systems (OSs) have the majority of the market share. In January of 2023, they were together responsible for about 99% of the available market.¹

To decide between mobile applications for Android or iOS (by Apple) we took several aspects into consideration. As discussed in the pre-project [84], we found that the Android OS has about 73% of the market share within mobile devices today [32]. In addition, there are more security issues related to the Android system than the iOS system. This is particularly regarding application permissions, as these are more easily manipulated in Android applications. An attacker can force the user to accept invasive permissions when opening an app on an Android device. These applications can also interact with the entire operating system in a more widespread way than in iOS, which leads to an attacker having the possibility to do more damage [32].

For our thesis, it was most relevant to look into previous work done within the field of medical application security. These applications contain features such as

¹<https://gs.statcounter.com/os-market-share/mobile/worldwide>

disease prevention, treatment, diagnosis, and patient monitoring. In addition, there exist multiple applications that are dedicated to maintaining good health for the general public. An example is mobile applications connected to smartwatches, that can give activity level, sleep information, and other general health facts to the user based on the sensor data from the watch. Within IMDs there seems to be a focus on the devices themselves, and not on the applications, within the existing literature. This relates to the articles discussed in the section about previous work on pacemaker hacking in section 2.2.

Previous research in the field of medical application security has also explored applications connected to other healthcare devices. In our pre-project, we mentioned two relevant articles on this topic, which both discussed security and privacy issues for mobile healthcare applications [84]. The first article composed a set of rules to follow when developing secure healthcare applications [44], while the second focused the discussion on possible attacks on healthcare applications, and the consequences these would have [56].

Another relevant article is the one from Sun et al. [78], discussing the security and privacy of the Internet of Medical Things (IoMT) enabled healthcare system. This article gives an overview of the security and privacy challenges and requirements for IoMT. Particularly, the security requirements part relates to our thesis, as it deals with conditions for both the data, the device itself, and the personal gateway.

Balapour et al. wrote an article on the role of perceived privacy as the predictor of security perceptions within mobile applications [9]. The focus was on the possible mitigation of security violations, through the improvement of the security behavior of users. They looked at the difference between the perception of privacy and the perception of security. In particular, the authors studied how the perceived security of a mobile application was influenced by the users' perception of privacy risks. They found that the perceived privacy of an application directly impacts the users' perception of the security of that application. This is valuable for us when we are looking into the mobile applications for IMDs.

The paper by Weichbroth and Lysik identified and analyzed existing threats and best practices within the domain of mobile security [86]. The authors used their collected data to perform a survey among mobile application users, regarding their awareness of threats and personal countermeasures. After a run-through of several different threats to today's mobile devices, the paper introduces concrete mitigation suggestions and discusses best practices. The study conducted concluded that the respondents had good knowledge of the risks and potential threats connected to mobile applications, but the personal countermeasures among the users were lacking.

Within the field of applications related to healthcare, the article by Llorens-

Vernet and Miró addressed the available standards and their criteria [46]. From there, a guide was developed with important criteria for designing a health-related application. In addition, this guide can be used when measuring the quality of a healthcare mobile application. The contribution of the paper was collecting criteria from several different sources, analyzing these, and making a common set of criteria and categories to be used by medical personnel, developers, and testers of mobile applications. This guide is relevant for us in our analysis of mobile applications.

2.2 Previous Work on Pacemaker Hacking

There is a growing awareness of security breaches within the field of IMD. Since these medical devices can be life-critical for the users, they can be a target for different groups of attackers. An exploration of the previous work within this field was conducted in our pre-project [84]. No new findings were discovered in this thesis, so the findings from the pre-project are presented in the following sections.

Researchers have written several papers regarding the security vulnerabilities in IMDs, where the first paper investigating the cybersecurity of an IMD was published already back in 2008 by Halperin et al. [33]. This research included an investigation of the exploitation of vulnerabilities of ICD, with a software radio-based attack to impact the patient's safety and privacy. It showed that the ICD was vulnerable to unauthorized tampering and could be threatened by attackers with low-cost equipment. This security breach was found by using reverse engineering, which was executed on the protocols on the communication link between the ICD and the ICD programmer [33]. Since its release, this paper has been the foundation for later contributions within IMD security.

In 2010, Denning et al. conducted a study to gain insights into the patient's perspective regarding embedded devices. The researchers conducted interviews with patients who had an implanted wireless IMD [24]. The interviews highlighted the importance of involving both patients and doctors in developing security mechanisms for future IMDs. During the study, participants were told to use mockups of the IMD security systems to envision future systems. Through interviews, they were able to identify the advantages and disadvantages of these systems. Their contribution was to highlight how the patient's perspective can interact with the technical properties when designing new and improved versions and systems. Furthermore, they discussed multiple approaches to increase the security of the IMDs.

In 2017, Zheng et. al analyzes security issues in IMD, which covered security trade-offs and the lack of security in the design of these devices [94]. It exists two different access modes, one for emergencies, and another for normal functioning, which results in a challenge in the security of the medical device. One of the trade-offs

demonstrates the importance of security vs. accessibility. The IMD should have a security mechanism implemented, while at the same time providing authorized personnel the possibility to access the system in case of emergency incidents or regular health check-ups. The emergency mode should not hinder the doctor to access the device in case of check-ups. The researchers also contribute with a security design, a biometric-based security scheme to secure the IMDs and to give access to doctors by using the patient's biometrics.

In the following years, different parts of the pacemaker ecosystem have been researched individually. According to a 2016 report by Muddy Waters Capital LLC, conducted in collaboration with the cybersecurity research company MedSec, pacemakers and heart monitoring equipment produced by St. Jude Medical have been found to be susceptible to attacks [45]. The pacemakers made by St. Jude Medical were vulnerable to cyber attacks due to security flaws in the software and hardware design. More specifically, the pacemakers had security vulnerabilities that could allow hackers to remotely control the devices, cause them to malfunction, drain the battery, or deliver inappropriate shocks to the patient's heart. The vulnerabilities were believed to be a result of poor design choices and insufficient security testing. An attack was performed and caused a significant risk to the patient's life, compromising both encryption and authentication in the communication link.

Kristiansen and Wilhelmsen's 2018 master thesis [93] addressed the pacemaker programmer from Biotronik. Their contribution was discovering violations against the CIA triad, particularly the lack of authentication for the programmer and other users, leading to security breaches compromising patient data confidentiality. This thesis was the first one published at NTNU in collaboration with SINTEF within the research field of the pacemaker ecosystem by Biotronik. Since, there have been five master theses from 2018 to 2022, all with slightly different focus areas.

In 2019, Bour [13] and Lie [43] investigated Biotronik's versions of the HMU in the pacemaker ecosystem. They both analyzed the security of the Cardiomessenger II-S, but different parts of the ecosystem. They successfully detected vulnerabilities impacting the patient's safety and privacy. Bour confirmed the hypothesis that an attacker having physical access impacts the security of an HMU device.

Later, a newer version of the HMU by Biotronik was tested by Kok and Markussen in 2020 to see if the security had improved from the previous versions and to uncover potential security vulnerabilities. The research resulted in finding breaches in the HMU by developing their own fuzzing framework targeting the SMS interface of the device [39]. The confirmed hypothesis on the old version of the HMU in Bour's thesis was to some extent also applicable to the latest version. The findings from the HMU analyses were later used to compare with the new system of the pacemaker ecosystem.

Chapter 3

Technical Background

This section presents technical background that is useful for the understanding of the rest of the thesis. We introduce guidelines and standards that form the foundation for our analysis further on. In addition, we present some security terminology to provide definitions of applicable terms. Lastly, we discuss how an Android application is constructed.

3.1 Relevant Guidelines

For our thesis, we have looked into two different guides, the OWASP Mobile Application Security Testing Guide (MASTG) and MITRE's Playbook for Threat Modeling Medical Devices. These are relevant to reach the goals of the analysis and to find risk mitigation strategies. These guides have helped us acquire knowledge to execute the threat modeling and analysis of mobile applications. We use both of them as a basis for our further work.

OWASP MASTG is the first guide, and this concerns the testing of applications. This guide covers the processes, techniques, and tools used during the analysis of mobile application security. It is based on verification of the requirements in OWASP Mobile Application Security Verification Standard (MASVS) and includes test cases for these. The aim is to provide a baseline for complete and consistent security tests [68]. The MASVS defines a security model and a list of generic security requirements for mobile apps.

The MASTG is split in several parts. First, they discuss some key principles of mobile testing; going through the terminology, different types of testing, and strategies. The guide focuses both on the security of the application and the privacy protection of the user. In addition, it includes dedicated chapters for mobile application testing for Android. These include an overview of Android, as well as useful tools and starting points for security analysis.

MITRE's Playbook for Threat Modeling Medical Devices is the next helpful resource for our thesis. It offers insights into how threat modeling is applied in organizations in the medical device industry. It presents a definition of threat modeling and how the analysis works, which is also described in chapter 4. A general approach to a threat model of a fictional medical device is included, presented with its threats and corresponding mitigations [14]. This playbook is used as a starting point in the development of a threat model for the new version of the pacemaker.

3.2 Standards

Some standards for security vulnerabilities were necessary for our security analysis. These are collections of vulnerabilities that are used by developers to increase the security of their applications. The three standards we discuss in this part are widely recognized among the security community. In the threat modeling part of the thesis, some flaws are connected to these standards.

OWASP Top 10 is an awareness document for web and mobile application security. This standard represents a broad consensus about the most critical security risks to web applications [30]. The Top 10 mobile risks from 2016 are considered in our investigation and are taken from OWASP's webpage [83]:

M1: Improper Platform Usage: focuses on the misuse of features or failure of the platform. This can lead to accessing security controls like Android intents, permissions, misuse of TouchId, the Keychain, etc.

M2: Insecure Data Storage: includes accessing the device's filesystem and data storage. This can for instance result in data loss, accessing sensitive data, fraud, and reputation damage.

M3: Insecure Communication: means exposing sensitive data on the device's network traffic when transmitting it. This happens when SSL/TLS is only used during authentication and not elsewhere and therefore risks phishing and Man-in-the-middle (MITM) attacks. This violates the user's confidentiality resulting in theft, fraud, or reputational damage.

M4: Insecure Authentication: can give an adversary the possibility to fake or bypass authentication or other interactions with the mobile application. Poor authentication leads to an inability to detect the identity of the user performing an action, the source of an attack, and prevent attacks in the future.

M5: Insufficient Cryptography: focus on improper encryption which can lead to sensitive data exposure or system compromise.

M6: Insecure Authorization: can cause an adversary to understand the authorization scheme and pass the authentication scheme, resulting in accessing an

application as a legitimate user. This weakness can also lead to obtaining sensitive information or compromising the system.

M7: Client Code Quality: relates to poor code quality and can be exploited by static analysis or fuzzing. The adversary can identify, for instance, memory leaks, and buffer overflows. Even low-skilled hackers can exploit these vulnerabilities.

M8: Code Tampering: involves code modification in a way that an adversary changes the intended use of the application. This results in a malicious, unauthorized mobile application either from a third-party app store or via phishing attacks, on the user's device.

M9: Reverse Engineering: involves an attacker analyzing the targeted application and deconstructing it to uncover its functionality and underlying mechanisms. This may result in exposing information about back-end servers and performing attacks against them, or gaining necessary information to do code modifications.

M10: Extraneous Functionality: includes an attacker looking for extra features on the mobile application to find potential hidden functions in the backend systems to perform an attack. This can give the attacker unauthorized privileges or knowledge of how the backend systems operate.

OWASP MASVS is another standard for mobile application security. It can be used either by developers to produce secure apps, or by security testers to ensure completeness of tests. This is used when we test mobile applications on Android to ensure that the test results are accurate, consistent, and comprehensive [62]. We evaluate and compare the security of mobile applications against this standard.

CWE stands for Common Weakness Enumeration and provides a list of weaknesses and vulnerabilities concerning both software and hardware. The standard is operated by the MITRE Corporation [20]. In addition, it includes mitigation and prevention methods. It consists of over 600 categories, divided into different weaknesses classes, like permission issues, random number issues, key management errors, authentication errors, etc.

3.3 Security Terminology

This section defines the necessary terms in cybersecurity to give a comprehensive understanding of the rest of the thesis. The well-known information security model used for guiding and development includes the triad *Confidentiality, Integrity and Availability (CIA)*. This was the core foundation of security systems and organizations, and they interlink with both *Authentication* and *Non-repudiation*. In our pre-project [84], we already discussed the terms and are repeating them below for clarity. All the definitions were gathered from National Institute of Standards and Technology (NIST) Computer Security Resource Center [57]. More specifically they are gathered

from FIPS 200, based on *44 U.S.C., Section 3542*, where the different terms have their own sections.

Confidentiality in security is defined as “preserving authorized restrictions on access and disclosure, including means for protecting personal privacy and proprietary information” by Federal Information Security Management Act (FISMA). In other words, the term’s meaning is to ensure that data is kept private. This implies that it should not be possible for unauthorized people to access the data of the application.

Integrity is defined as “guarding against improper information modification or destruction and includes ensuring information nonrepudiation and authenticity” according to FISMA. This is often related to the data of the application and concerns its correctness. This means that it is not possible for someone to alter, delete or add data without this being discovered.

Availability means “ensuring timely and reliable access to and use of information”, also defined by FISMA. It concerns the service as a whole, where the application should be possible to access for authorized users whenever they wish to do so.

Authentication is described as “verifying the identity of a user, process, or device, often as a prerequisite to allowing access to resources in an information system” in Federal Information Processing Standards (FIPS) developed by NIST. Thus, it is the process of making sure that a user of the service is who they claim to be and that this authenticated user is allowed to access the service.

Non-Repudiation is the “assurance that the sender of information is provided with proof of delivery and the recipient is provided with proof of the sender’s identity, so neither can later deny having processed the information” specified by NIST. The definition implies making sure that it is not possible to deny the validity. For example, it should not be possible for a sender to deny that he is the source of a packet.

These concepts were applied in the context of the pacemaker ecosystem. Here are some examples of security breaches for each of these terms, where an unauthorized and/or non-medical third party and/or malicious actor are referred to as an *unauthorized entity* using the Playbook [14] as a guide:

Confidentiality: when private patient information or data is disclosed by an unauthorized entity without the patient’s knowledge.

Integrity: when an unauthorized entity tampered with and modified the patient data into incorrect information, which is later used for a healthcare check-up.

Availability: when blocking access in the communication link between the HMU and the backend server which prevents legitimate activity, resulting in patient data information being unavailable for a time period.

Authentication: when an unauthorized entity finds a way to cheat into the system as a medical professional, which can be done by stealing login credentials from authorized personnel.

Non-Repudiation: when logging into the system as an unauthorized entity, but pretending to be a medical professional, and later denying having done this malicious event.

In the healthcare system, one of the main stakeholders is the patients. This group is highly considered in this thesis' analysis, focusing on protecting the patient's privacy and safety. As a consequence, it is essential to clarify the difference between *Security*, *Safety*, and *Privacy* which are terms with similar meanings but used differently. The definitions are gathered from the Computer Security Resource Center by NIST [57].

Security has a wide meaning and has plenty of definitions presented. NIST defines the term as “freedom from those conditions that can cause a loss of assets with unacceptable consequences.”¹ An additional, and more informative definition of security is “protecting information and information systems from unauthorized access, use, disclosure, disruption, modification, or destruction in order to provide integrity (guarding against improper information modification/destruction, includes ensuring information non-repudiation and authenticity), confidentiality (preserving authorized restrictions on access and disclosure, including protecting the privacy and proprietary information), and availability (ensuring timely and reliable access to and use of information).”²

Safety is described with two definitions, where the first states “freedom from conditions that can cause death, injury, occupational illness, damage to or loss of equipment or property, or damage to the environment.”¹ The next definition says “expectation that a system does not, under defined conditions, lead to a state in which human life, health, property, or the environment is endangered”.²

Privacy is defined as “the assurance that the confidentiality of, and access to, certain information about an entity is protected.”³ Further, privacy is also defined as “freedom from intrusion into the private life or affairs of an individual when that intrusion results from undue or illegal gathering and use of data about that individual.”⁴

¹NIST SP 800-160 Vol. 2 Rev. 1

²NIST SP 800-66 Rev. 1 under Security from 44 U.S.C., Sec. 3542

³NIST SP 1800-10B under Privacy from NIST SP 800-130

⁴NISTIR 8053 from ISO/IEC 2382

Other significant terms in the cybersecurity field that are vital to make sure are defined correctly are *Risk*, *Threat*, and *Vulnerability*. Despite the fact that the terms are used interchangeably and often blend together, they have different definitions. These were used in the process of threat modeling in section 4.1, to give a better and deeper understanding of how an organization can optimize security. These definitions are proposed, similar to the definitions described above, in the glossary from NIST’s cybersecurity- and privacy-related publications [57]. More specifically they are gathered from NIST SP 1800-15B, where the terms have their own sections.

Risk is defined as “a measure of the extent to which an entity is threatened by a potential circumstance or event, and typically a function of (i) the adverse impacts that would arise if the circumstance or event occurs; and (ii) the likelihood of occurrence.”

Threat is described as “any circumstance or event with the potential to adversely impact organizational operations (including mission, functions, image, or reputation), organizational assets, or individuals through an information system via unauthorized access, destruction, disclosure, modification of information, and/or denial of service. Also, the potential for a threat source to successfully exploit a particular information system vulnerability”.

Vulnerability is defined as “a weakness in an information system, system security procedures, internal controls, or implementation that could be exploited or triggered by a threat source”. Further, vulnerability is also defined as “a flaw or weakness that may allow harm to occur to an IT system or activity”.

3.4 Android Applications

The analysis of applications performed in this thesis was based on applications for the Android operating system. In order to understand how the analysis took place, we need to look into how Android applications are built. For our analysis, we used Android Package Kit (APK) files, which is a file format used by the Android operating system to install and distribute applications. Every application that a user has on their Android phone can be downloaded as an APK file. This APK file is a version of a ZIP format and contains all the elements and files necessary for a correct installation on the device. Some of the files contained in these APK files were relevant to our analysis, and we introduce their purpose here.

One of the most important files in the APK, is the *AndroidManifest.xml*. This is an obligatory file for all Android applications, and it includes important information about the app. First, the manifest includes all the different components of the application, such as activities, services, broadcast receivers, and content providers.

Each component must declare the name of its class, and can also declare capabilities. In addition, this file includes the different permissions the application needs from the OS of the phone to access protected parts of the system. The permissions needed for other apps to access the content from this application are also declared. Lastly, the manifest file includes information about which hardware and software feature the app supports, which can limit which devices that can download and use it.

In addition to the manifest file, an APK file consists of other parts. There is a directory called *assets* containing all the applications assets, a *lib* directory that includes compiled native libraries used by the app, and a *META-INF* directory with APK metadata, such as the signature. In addition, there is a *resources.arsc* file with precompiled resources, such as strings and styles, and a *res* directory with all resources that are not compiled into the file. Lastly, the APK contains a *classes.dex* file with all the application code in Dex file format, which is a Dalvik executable format, used to enable the execution of the code in a virtual machine [79]. For our analysis, we look into both the code itself, the shared libraries used in the application, and the certificates in use. In particular, the *classes.dex* file is important, as we need it to gain an understanding of the source code of the application.

Chapter 4

Methodology

The primary objective of this thesis is to do a security assessment of the new evolution of the pacemaker ecosystem, with the aim of comparing its security measures with those of previous versions. To develop an understanding of the threat landscape and risks, we perform a threat modeling of the system. Similarly, to identify the technical vulnerabilities of a mobile application, black box testing was applied to analyze relevant healthcare mobile applications. Different tools were used to analyze the security level of these. In addition, we conducted a survey to gain insight into a patient's perspective.

4.1 Threat Modeling

The threat modeling method was applied to discover the security threats concerning the patient's safety and privacy. The method was used to identify potential threats that could have an impact on the pacemaker ecosystem, targeted by a malicious attacker in order to demand a high ransom, harm, or kill patients if the system is compromised. Attackers are aware of the value of patient data, thus exploiting vulnerabilities in these systems can be highly profitable. Another motivation can be targeting a specific person. To protect the security of the pacemaker's new digital solution, it is important to have a threat model that provides a structured representation of relevant information. This model provides information and helps make educated decisions regarding application security. This also identifies and improves countermeasures to prevent or mitigate potential threats.

A threat model includes a description of the system to be modeled and potential threats to the system. In addition, mitigation techniques and actions related to the threats were discussed. To organize the threat modeling and to provide different insights into the pacemaker system, four key questions are presented, provided by MITRE Playbook for Threat Modeling [14]:

1. What are we working on?
2. What can go wrong?
3. What are we going to do about it?
4. Did we do a good enough job?

In this thesis, the main focus was questions 1 and 2. To answer question 3 we provided potential solutions, while the last question was barely touched. This was because the process of threat modeling is a continuous activity, whereas we only analyzed this system without applying any of the proposed improvements from question 3. This meant that it would be difficult to assess if the improved system was adequately secure, as we would not reiterate over a new version of the system.

4.1.1 Datagram Flow Diagrams

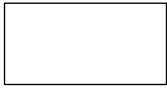

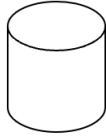
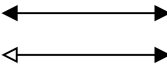

The first step of threat modeling required visualization of the system before we could dive further into the process. To answer question one, *what are we working on?*, we needed a good understanding of the system and build a model to get a structured approach. We adopted this approach to achieve an overview of the security gaps in the system and for easier knowledge sharing for the system's stakeholders. To structure and visualize the pacemaker system, we decided to use DFDs that represent the entities and their interactions and relations. The modeling with DFD included DFD objects, where icons were used to represent different elements, shown in Table 4.1.

To start with developing the diagram, an identification of the major components of the pacemaker ecosystem was needed to get a high-level overview of the system. Then, a deep dive into the system was done to add more detailed and relevant entities to the diagram, resulting in a mid-level diagram of the system. The external entity describes the entities that are outside of our control, the processes represent any running code, the database symbolizes where data is stored, and the communication is in the data flows. In contrast to these physical components, the trust boundaries represent the trust levels between different objects and describe the interaction between them. By defining these, we achieve a clear picture of which parts of the system needs to be studied further.

4.1.2 STRIDE

There exist multiple approaches to find out what can go wrong in the system. This stage focused on identifying and documenting any mitigation controls associated with the threats. Additionally, we had to brainstorm what could go wrong if these

Table 4.1: DFD objects

| Element | Symbol | Description |
|------------------|---|---|
| External entity |  | A sharp-cornered rectangle |
| Process |  | A round-cornered rectangle |
| Database |  | A drum |
| Data flow |  | Double-headed arrows. Empty arrow-head means that the communication is only initiated by one side |
| Trust boundaries |  | A dashed rectangle |

controls were to fail. Despite the fact that control mechanisms exist and were present, it does not mean automatically that the system was not vulnerable. Therefore, it was crucial when threat modeling to develop an understanding of the potential threats to enhance resilience.

STRIDE stands for Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, and Elevation of privilege. This methodology was used for identifying security threats and provided a foundation for identifying threats. It is a mnemonic for six categories of threats. Each category is mapped to a description and one security property described in Table 4.2 with guidance by the MITRE Playbook [14].

Applying this method in the pacemaker system could help us find an answer to the second question in the threat modeling, *what can go wrong?*, within these categories. A table with brief information about each STRIDE element with an associated description and an example within the pacemaker ecosystem was provided. Subsequently, the STRIDE model was applied to the ecosystem's components. We used DFD to see which elements were affected by which threats and document these to possibly find mitigation methods against these later on. To do this, a method called

Table 4.2: Description of the STRIDE elements with the associated security properties

| Element | Description | Property |
|------------------------------|---|-----------------|
| Spoofting | Tricking a system into believing a falsified entity is a true entity and gaining illegal access to privileged data | Authentication |
| Tampering | Intentional malicious modification of information and resources of a system in an unauthorized way | Integrity |
| Repudiation | Disputing the authenticity of an action taken without other parties having a possibility to prove otherwise | Non repudiation |
| Information Disclosure | Exposure and illegitimate availability of information to malicious and unauthorized users intended to have restricted access levels | Confidentiality |
| Denial of Service (DoS) | Blocking legitimate access or functionality of a system by malicious process(es) causing service unavailability for authorized, legitimate users or programs | Availability |
| Elevation of Privilege (EoP) | Gaining access to functions to which an attacker should not normally have access according to the intended security policy of the product, and thereby having sufficient access to compromise or destroy the system | Authorization |

“STRIDE per Element” was executed to pair a threat to a particular DFD element to investigate threats that cross trust boundaries in the pacemaker ecosystem. A more detailed summary table was provided in this step, which connected the components in the pacemaker with one or more identified threats.

4.1.3 Mitigation Techniques

When we discovered a threat, we focused on how to manage it by finding defensive techniques. In threat modeling, there are four strategies for addressing threats - eliminating, mitigating, accepting, or transferring. When using the eliminate approach, the goal is to identify and eliminate features that the threat is applicable to. An example is stopping the collection of specific patient data that is not necessary, so it can not possibly be stolen. When accepting the risk of a discovered threat, the organization accepts and acknowledges the potential loss that is not necessary to spend more resources and money on. It is difficult for us to decide which risks

that are acceptable according to the organization's values. The transfer approach means that the risk is transferred to a third party, usually the manufacturer. This is typically achieved through license agreements and terms of service, which outline the responsibilities of both parties with regard to the product's potential risks and any harm that may result from its use.

In the next section, we focused on the mitigating approach which helped us answer the third question of threat modeling, *what are we going to do about it?*. According to the MITRE Playbook for Threat Modeling, a threat is a possibility of a negative event happening, while a risk is more complex and includes the probability of that event happening, as well as the potential impact if it does happen [14]. As explained in section 3.3, to clarify the terms threat and risk regarding question 3, *what are we going to do about it?* can be better understood as “how do we manage the risk from the threats we have identified?” [14].

The goals of mitigating threats were to identify which control mechanisms to add with the purpose of stopping potential attacks and also to improve these mechanisms. The system is evolving continuously which makes it necessary to control the system against vulnerabilities when different configurations and features are changed or added. In consequence, having the documentation of fundamental information about threats and associated mitigation techniques was essential for future threat modeling of the system. The mitigation controls are often divided into functions which are primary pillars to guide organizations to manage security risks at a high level and do proper risk management decisions. The controls are divided into four categories according to Cybersecurity Framework by NIST [22]:

Protect includes developing and implementing safeguards to limit the impact.

Protection control has the intention of protecting patients' safety and privacy, to ensure that critical services for a patient are safely delivered.

Detect is described as identifying and detecting unnatural and unwanted behavior of the system. This includes developing relevant activities to discover occurrences like these. Detection control has the ability to perform immediate action to prevent and minimize the impact of an attacker.

Respond is to take action when a security incident is detected and react as soon as possible. In the process of responding and knowing how to react to different scenarios, there is a need for activities showing how to take action in case of a security breach.

Recover controls have the purpose of maintaining resilience. The intention is to return back to a state where additional security mechanisms are not necessary to be added to counteract the threats. Afterward, checking if the system is

working correctly may be needed for further service to the patients as a part of the recovery controls.

4.2 Black Box Testing

After the execution of the threat modeling of the system, two medical healthcare applications were chosen, and we performed an analysis of these. When it comes to testing mobile applications, there were several methods available, depending on the amount of information the tester has about the system.

To describe different versions of software testing, the imagery used is often a box. This box has different colors depending on the amount of knowledge the tester has of the internal workings of the system. Firstly, there is white box testing, which is when the tester has full transparency of the system. This means that the tester has access to the source code and all other documentation associated with the software. This leads to better test coverage since all the code can be tested. This type of testing is often executed before the software is published or used. White box testing can be conducted by the authors of the software, or others within the same organization, or the company can bring in external testers to gain a more objective look at the security.

On the other side of the spectrum is black box testing. In this case, the software is seen as a black box, which means it is unknown to the tester. In practice, this means that the tester must rely on publically available information to gain knowledge of the app. The tester does not know the internals of the software and interacts with the application to validate how it works and whether it meets some requirements. The tester defines test cases and uses the application as a user, to see if the software behaves as expected. This type of test can lead to some vulnerabilities going unnoticed, as the tester cannot access all the source code. Black box testing can also be more time-consuming to execute, as the tester does not know how things are connected, which can lead to a tedious job of setting up the tests. On the other hand, the black box approach is very valuable as a test method, as it is the most similar to real-life scenarios with actual attackers.

In addition, grey box testing also exists, which is a common name for all types of testing that falls between white and black box. This means that the tester has some knowledge of the system's internal workings. For example, the tester can be given access to an authenticated user profile and, therefore, is able to explore all the features of the application. This can help to guide the evaluation of the system. Usually, the tester does not have access to the source code in this case but can have little knowledge, such as how the application is structured. The internals relevant to the testing can be given to the tester. This can provide a better variety of test cases

and is less time-consuming to execute for the tester.

In this thesis, the type of testing performed was black-box testing. Our primary goal was to find the security vulnerabilities in mobile applications and utilize these. We had no prior knowledge of the applications' internal workings, hence black-box. On the other hand, testing an Android application with access to its APK file and its source code can be considered a form of gray-box testing. We had a limited understanding of the applications' code structure, design, and architecture. However, as testers, we did not have completely authorized access to the internal processes and development of the application. To conclude, the testing executed was a combination of both gray- and black-boxing in our opinion. The details of how we performed the analysis of the applications are further explained in the next section.

4.3 Analysis of Applications

The first step in executing the analysis of healthcare applications was to search for suitable apps. The goal was to stay within the field of applications connected to pacemakers. Therefore, we looked into several manufacturers of pacemakers to see if they provided the option to choose an IMDs with a connected application. We visited the manufacturers' websites and looked for public information here, as well as searched through application stores for Apple and Android. Applications discovered are represented in Table 4.3. There were applications offered by Medtronic, Biotronik, Boston Scientific, and Abbott. For our study, we focused on two of these applications, as we did not have the time to analyze them all. To make the analysis encompass as much as possible, we chose one application of each type - one that substitutes the HMU and one that is an addition to the system for patients. The previous master theses written on the topic of the pacemaker ecosystem, presented in section 2.2, had focused on Biotronik's ecosystem. Therefore, we found it logical to study their application for our analysis, to make the comparison between the older and newer version of the system more accurate. When it comes to the application that substitutes the HMU, we decided to base our analysis on Medtronic, as it is an industry leader within the field of pacemakers. These two applications were the basis for our security analysis.

4.3.1 Extracting APK Files

After deciding which applications to use, APK files of these had to be extracted. Mobile applications for use on Android devices have publically available APK files. They are usually downloaded via Google Play Store but are also available on other websites. If we wish to open the application on our desktop, we would need to use an emulator as it is not possible to open an APK file directly. We chose the Android Emulator that came with Android Studio. To find out how to create and run an

Table 4.3: Available applications from different manufacturers

| Manufacturer | App name | Role | App Store | Google Play |
|-------------------|------------------------|-----------------------------|-----------|-------------|
| Medtronic | MyCareLink Heart App | Substitute for hardware HMU | Yes | Yes |
| Biotronik | Patient App | Addition for patients | Yes | Yes |
| Boston Scientific | MyLATITUDE Patient App | Addition for patients | Yes | Yes |
| Abbott | MyMerlin App | Substitute for hardware HMU | Yes | Yes |

emulator, see appendix A.2. The APK was dragged directly into the emulator to be downloaded on the emulator. The APK file itself can be extracted from the application on an Android phone, by using a program such as APK Extractor.¹ After extraction, the content of the APK folder can be shown by unzipping the file.

4.3.2 Decompile APK File to JAR File

To further understand how the application worked, a tool called *apktool* [5] was used to decompile the APK file. This made a folder with different files and directories, which we described in more detail in section 3.4. For the code analysis, it is the *classes.dex* file that was of the most importance. Since this file was in a Dalvik EXecutable (DEX) format, suitable for the Dalvik machine, we had to export this file into a more readable format. This could be done by the *dex2jar*[64] tool, which transformed DEX files into Java ARchive (JAR). This was a collection of Java files and the code that was used to build the application. We used this to identify potential security vulnerabilities.

When the file was transformed into a JAR file, it was possible to look through the code. To open the file, we used the program *jd-gui*[37]. This is a Java decompiler, an editor that allows us to see through the code, observe its structure, and make changes. Further instructions on how to use the tools can be found in appendix A.1.

4.3.3 Tools for Security Analysis

After extracting the APK file and opening this on our computers, we used several different tools for the analysis of the applications. Our goal with the analysis was to use publically available tools to discover unexpected findings without using very

¹APK Extractor: https://play.google.com/store/apps/details?id=com.ext.ui&hl=en_US&gl=US&pli=1

advanced instruments. The reason for this was to study how a low- to medium-skilled attacker could be able to gain insight into the applications. In order to accomplish this, relevant tools were searched for, that offered an off-the-shelf analysis of applications and were free to use or low-cost. This gave the most realistic view of the vulnerabilities that exist within the applications.

We used the reverse engineering technique, which covers a broad range of areas. This included decompiling and disassembling executable files and libraries, and analysis of system data. The primary goal of reverse engineering is to understand how a system works, its vulnerabilities, and the protection mechanisms against them. In the context of mobile applications, reverse engineering includes deconstructing, analyzing, and observing compiled applications. This gave an understanding of the application's code, logic, and functions. There are two main types of reverse engineering, namely *Static Analysis* and *Dynamic Analysis*.

First off, we executed a static analysis of the applications to obtain an initial, general, and high-level understanding. With static analysis, the application is inspected without running it. Next, a dynamic analysis was performed to achieve a more detailed application knowledge. In this process, the application was analyzed during runtime. In this type of code analysis, we could obtain actual values at runtime and a network traffic overview. As a result, some additional security vulnerabilities may be available in dynamic analysis vs. static analysis. In addition, running the program in dynamic analysis can cause overhead in terms of time and resources and can slow down the execution. These are just a few pros and cons of each code analysis.

Static Analysis

To perform the static analysis of the mobile applications, we used two different tools, available for free. The first one was MobSF, which is an automated framework for executing penetration testing, malware analysis, and security assessments of Android or iOS applications. MobSF worked well for executing quick security tests of the application, especially during development. The framework has a Graphical User Interface (GUI) that represent and visualizes the results, which made it easier for us as researchers to have a clear overview of the information displayed. This included permissions, APIs used, and multiple analyses concerning certificates, the manifest, the code, potential malicious behavior, etc. MobSF provided us with an initial overview of potential vulnerabilities in the application and gave us a starting point for further investigation.

MobSF gives an overall security score to the applications that are analyzed. This score is calculated by giving each app an ideal score of 100, to begin with, and then reducing or adding to the score depending on the findings. For a finding with severity

high the score is reduced by 15, while for a finding with severity *medium* the score is reduced by 10. On the other hand, if there are findings with severity *good*, 5 points are added to the score. Based on the summation of these calculations, the applications are assigned a risk category. This is distributed as follows, given by [Risk category: Security score range]:

- **Low:** 71-100
- **Medium:** 41-70
- **High:** 16-40
- **Critical:** 0-15

A MobSF analysis divides the result into sections, and the threats were labeled in different categories, depending on the severity level. The severity levels are *Secure*, *Info*, *Warning*, and *High* depending on the grade of the threat's malicious behavior. The level High is given to issues that are a direct threat and can imply malicious behavior, while Warning is associated with issues that can be a vulnerability, depending on the implementation. Issues within the Info category are things that the developer should be aware of, but that do not pose a direct threat to the application. The category Secure is given to issues that have no connected security vulnerabilities. Moreover, the permissions do not have severity levels, but rather a status. These are distinguished between *Normal*, *Dangerous*, *Signature*, and *Special*. The type of permission depends on the extent to which the application accesses restricted data and performs restricted actions.

The other tool we used for the static analysis was BeVigil. This is a free static analyzer available online [11]. BeVigil gave a report for each app and the analysis consisted of many of the same categories as the MobSF analysis. BeVigil additionally provided the ability to see which files in the application were affected by an issue, and in some cases also looked into the specific code that the issue was gathered from. The website offered the ability to search for different applications, as well as upload your own APKs for analysis. Similarly, as for MobSF, the report had an initial security score and an elaboration of different issues within the categories of analysis. The score was given on a scale from 0 to 10. There was no specific information about the exact calculations of the score as it was only stated that the scores were calculated via a CVSS-based logic. In addition to giving a current score for the application, BeVigil provided three issues that the application could fix to improve its score in the most prominent way.

The BeVigil worked in a similar way as MobSF, where the vulnerabilities were split into different sections, with different severity level categories. The permissions

are ranged from *Normal* to *Risky* to *Dangerous*. Other issues in the analysis have severity levels in the range from *Low* to *Medium*, and further to *High*. We can thus see that the severity categories are a bit different from MobSF. However, these analysis tools are quite comparable.

Intercepting HTTP and HTTPS Traffic

Two proxy tools were used to act as HTTP/HTTPS proxies connected to an emulator. The first one was a proxy configured in Burp Suite. Burp Suite is a tool used for penetration testing of mobile applications [89]. The Burp Proxy was used to study and modify the content in the requests and responses while in transit, see appendix A.4 to figure out how to configure the emulator to an intercepting proxy. Mitmproxy was the next proxy for HTTPS. This is a man-in-the-middle proxy for debugging, testing, and penetration testing [54]. Appendix A.5 demonstrates how to configure the emulator to use a mitmproxy. With mitmproxy, you can create and execute a Python script to modify and send the responses back to the mobile to affect the behavior of the application. Both appendix A.6 and appendix A.7 illustrates how to intercept a request with the different tools. Additionally, in order to make a proxy work properly, it was essential that we could communicate with the emulator by using Android Debug Bridge (ADB) commands. See appendix A.3 for instructions on how to download ADB.

Frida

Frida is a tool for dynamic code instrumentation. It allows you to inject JavaScript into native applications during runtime. The tool can be used on applications running on Windows, macOS, Linux, iOS, Android, and others. It consists of a collection of built-in methods, to trace different functions being called during runtime, listing the available processes on the device, and discovering internal functions of a program. In addition, it provides a possibility to attach custom Python scripts to the running application, in order to override the functionality of the application dynamically. Frida can inspect functions as they are called, modify their arguments, and do custom calls to functions inside a target process [31]. We used the tool to attempt dynamically bypassing the authentication mechanism of the mobile applications in real-time. The process of setting up Frida is described in detail in appendix A.8.

4.4 Sending out Questionnaires

In order to gain valuable insight into the patient's perspective, we wanted to send out a questionnaire to patients with a pacemaker implanted. To find relevant respondents, we reached out to Facebook groups for pacemaker patients and asked the participants to answer our questionnaire. We wanted a patient group containing patients who

both used mobile applications as the gateway for pacemaker communication and patients who used the dedicated hardware HMU. This questionnaire resulted in a quantitative insight into the patient's viewpoint.

A questionnaire is a valuable tool to conduct surveys. The goal was to understand the topic from the respondent's perspective. For our thesis, questions regarding the patient's habits with security, their perception of privacy with regard to mobile applications, and other relevant behavioral questions were included. The questionnaire contained a variation of close-ended and open-ended questions, where the open-ended asked for short, written answers. In addition, we had to include some questions that asked the patients to answer along a Likert scale, from "strongly disagree" to "strongly agree".

4.4.1 Finding Questions

To keep the questionnaire anonymous, it was important to not collect personally identifiable information. The aim was to get some demographic information about the respondents, such as age and sex, in order to be able to compare the different demographic groups without getting enough data to identify individuals. These questions were the first part of the questionnaire. Further, we added some questions about which vendor the patient used and the type of device they had, before we moved on to more behavioral questions. For these, we used a combination of multiple-choice, closed questions with several answer options, and open-ended questions where the respondent was asked to write a short answer.

After some general questions, we asked some questions about their mobile phone use. This included what operative system they had, their habits regarding updates, and the security of their passwords. In this section, we asked whether the user was mindful of their presence online, and gained insight into how they secured their accounts. Thereafter, we moved on to questions about their perception of the applications connected to their device. First, we asked whether the patient used any app connected to their device, and asked for insight into what they perceived as benefits and disadvantages connected to the apps. We also asked for some information regarding how much the patients paid attention to the security of the applications they had on their phones.

The purpose of the questionnaire was to gather information about whether the patients used their mobile phones securely and their thoughts on the security of the healthcare applications they used. The patients' habits were gathered in order to see if there were added vulnerabilities to the system based on the routines of the respondents, for instance, if they rarely updated their phones or applications. The perceived security of the applications was gathered in order to see if there were any deviations between the actual security level of the applications and the perception of

the users, addressed in section 8.4. The full questionnaire sent to patients can be found in appendix B.

4.5 Ethical Concerns

There were a few ethical concerns that we needed to take into consideration for our master thesis. When it came to the questionnaire, we ensured the anonymity of the respondents by not collecting any personally identifiable information. Even though we ensured the non-collection of personal information, there are still important ethical aspects we considered in the questionnaire.

First, we obtained informed consent from the participants, explained who the questionnaire was meant for, its purpose, and what the responses will be used for. Furthermore, we used a survey platform that allows respondents to answer anonymously, ensuring that we, as collectors, were unable to associate the responses with specific individuals. The respondents participated voluntarily and none of the questions were mandatory. The alternative “Prefer not to answer” was provided in some questions. Lastly, the data is deleted at the end of the project. By addressing these ethical concerns, we aim to follow the principles of privacy, confidentiality, and respect for the participants involved in the questionnaire.

When it comes to our security analysis there were also potential ethical concerns. Before starting the thesis, we wanted to make sure to inform the vendors if any vulnerabilities were found. However, even though we found some issues in the security analysis of the applications, we did not find anything that introduced a high risk to the patient’s safety or privacy. Therefore, there are not any ethical issues with posting the results from this analysis before reporting them to the vendors. This also means that we do not need to postpone the publication of our thesis, as it is not necessary to give the vendors the opportunity to react to our findings and make improvements.

We also want to specify that during our security analysis, we performed ethical hacking. This means that we never interacted with the infrastructure from either Medtronic or Biotronik. We set up an emulator in order to facilitate the analysis of the application without communicating with the servers. Therefore, we intercepted all requests going out of the application. It was of high importance for us to make sure that the underlying infrastructure of the vendors was in no way harmed during our analysis.

Chapter 5

Threat Modeling

This chapter presents a threat modeling analysis of the updated pacemaker ecosystem. This helps us with answering our research question, described in section 1.4, by addressing research objective number 2 saying “*Identifying and analyzing different threats to this new version of the pacemaker ecosystem*”. The MITRE Playbook [14] served as a key tool in accomplishing this. The two versions of the current ecosystem can introduce different threats, and thus we perform threat modeling for each of these versions of the system separately. We take into consideration two ecosystems, the Medtronic ecosystem, and thereafter the Biotronik ecosystem. The Medtronic mobile app substitutes the HMU, while the Biotronik mobile app is an addition to the ecosystem and works in tandem with the HMU. As mentioned in section 4.1, there are four fundamental questions for this procedure, where the three first are the most relevant for us. Thus, these three are addressed in their own dedicated subsection for each vendor.

The attacks described in the threat modeling can be divided into active and passive attacks. A passive attack is an attack where the intruder simply observes the content of messages or stored information. On the other hand, an active attack is when the intruder tries to modify the content of a message or stored information [25]. A passive attack is a threat to confidentiality and the patient’s privacy, while an active attack exploits an integrity or availability vulnerability. This classification is important when it comes to the impact of the different threats, as active attacks can do more harm to the system, and thus can have larger consequences in contrast to passive attacks.

To clarify, our key focus is on the mobile application and the mobile phone. In other words, the remaining parts of the system are not as discussed, analyzed, and prioritized in the threat modeling process, but we still cover them briefly in order to get a more comprehensive look at the threats.

5.1 Medtronic’s Ecosystem

We performed a threat modeling of Medtronic’s pacemaker ecosystem. The goal was to identify potential vulnerabilities and threats, mitigate these, and protect patient safety. Firstly, it was important to understand the architecture and the design of the system. When the network components were defined, we could analyze the potential threats that could arise from these. Lastly, we provided several mitigation techniques to address the identified vulnerabilities.

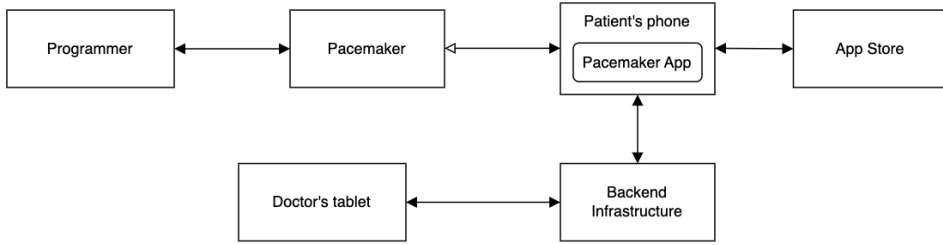
5.1.1 What Are We Working On?

In order to obtain a comprehensive understanding of the system, the initial step was to create a system model of the pacemaker ecosystem. As stated in section 4.1, DFD was used for constructing the system model. To start off, we identified the major objects of the ecosystem and continued to build out the DFD with more information on each individual object. We created a system model where the mobile application replaced the HMU. The system model for the Medtronic ecosystem can be seen at a high- and medium-level in Figure 5.1. For clarity, we referred to the device names under which Medtronic operates themselves.

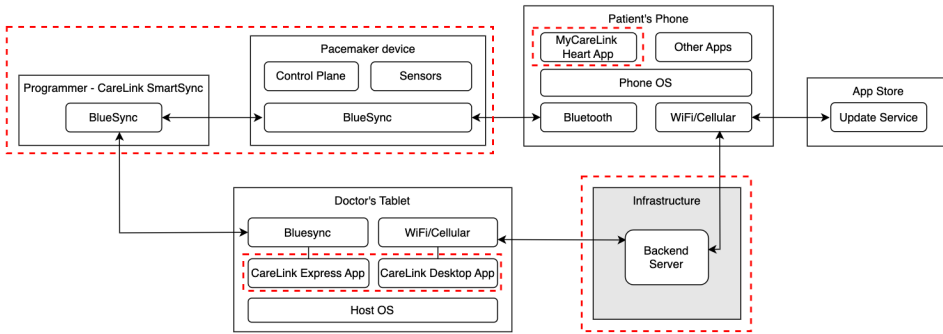
The high-level model includes the known devices that are part of the ecosystem. This model gives a brief overview of the different components, and how they are related. This high-level system model is presented in Figure 5.1a. Further, the medium-level model gives more insight into the inner workings of the different objects, illustrated in Figure 5.1b. All the information that is entered into the model has been found on Medtronic’s own websites, in the descriptions of their available products.¹ Therefore, there are some uncertainties on especially the backend of the system.

For the medium-level model, we have marked the infrastructure box in a grey color, representing a black box. Our understanding of its structure and functionality is limited. In fact, we only know that within this system there is some infrastructure that collects all patient data and that both the healthcare personnel and the mobile phone must be able to communicate with this infrastructure. The model also includes some trust boundaries, marked by the red dotted squares. These boundaries are meant to show which parts of the system reside within different levels of trust. In particular, the devices and services that Medtronic has control over have a different level of trust than external entities. All the devices within a red-dotted square should have control mechanisms that limit the interactions with the devices outside of the squares. The communication between the pacemaker and mobile phone within the

¹Medtronic Product Information. Available at: <https://www.medtronic.com/us-en/healthcare-professionals/products/cardiac-rhythm/managing-patients/bringing-it-all-together.html>



(a) High-level data flow diagram



(b) Mid-level data flow diagram

Figure 5.1: Data flow diagram of Medtronic's ecosystem

Medtronic trust boundary uses BlueSync, which is a protocol running on top of regular Bluetooth Low Energy (BLE). This protocol has enhanced security features, to ensure secure communication between the devices [50].

The DFD objects included in Medtronic's system model, and a short description of each, is illustrated in Table 5.1. Here we have both the physical components of the ecosystem, as well as the most important dataflows.

5.1.2 What Can Go Wrong?

To address question 2 in threat analysis, the STRIDE approach is applied to identify security threats to the pacemaker ecosystem, proposed by MITRE Playbook [14]. To start, a brainstorming session was executed, where all identified threats and attacks were written down. After this first round, we did a short literature study to fill the gap of missing threats. Further, a classification of threats was needed for clarity. Therefore, the DFD objects in the ecosystem received their own table with a description of each identified threat divided into one of the categories of threats in the STRIDE model. A summary of all the threats is presented in Table 5.8.

Table 5.1: Identified pacemaker components and their description

| DFD Object | Description |
|----------------------------------|--|
| Pacemaker device | This is the device situated inside the patient's chest, responsible for giving small electrical pulses to keep the heart functioning as normal |
| App | This is an Android application on the patient's phone. The app is responsible for collecting data from the pacemaker, showing some statistics to the user, and forwarding the data to the data server |
| Programmer | This is the device responsible for programming the pacemaker to function for each patient's specific needs. Resides at the hospital and is operated by healthcare personnel |
| Backend server | The server of the vendor collects and exports the data coming from the mobile phones of the patient. The data is accessible to healthcare personnel |
| Doctor's tablet | The computer/tablet belongs to the healthcare personnel. The tablet has an application with information about the different patients the doctor is responsible for. Can gather data from the programmer and the data server |
| Dataflow: Cellular/Wi-Fi Network | The data flow goes from the mobile phone of the patient to the data server, containing all the data from the pacemaker. The doctor can also gather data from the data server to their local computer/tablet over cellular/WiFi |
| Dataflow: Bluetooth | The dataflow going from the pacemaker to the phone. In addition, the data flow between the pacemaker and the programmer. Uses a protocol called BlueSync, running on top of the BLE protocol |

Each threat is associated with an impact, which is noted as low, medium, or high. The impact depends on the potential harmfulness to the functions, processes, and patients, and the number of patients affected. This is illustrated as a matrix in Figure 5.2. The green boxes represent low impact, where the threat has low consequences in which one or few patients are involved, and the threat causes little to no harm. Next, the yellow boxes indicate medium impact, where the threat has moderate consequences. These threats are more significant than low-impact threats but not as severe as high-impact threats. They are not urgent but are important and require attention. Lastly, the red boxes denote high impact, where the outcome has a high effect on a large group of patients or is life-threatening to a patient. Additionally, each threat was categorized as resulting in a passive or active attack, based on the required approach from an attacker. In the tables, we are listing the threats as either P, for *Passive*, or A, for *Active*.

Table 5.2 summarizes the brainstormed threats related to the pacemaker device,

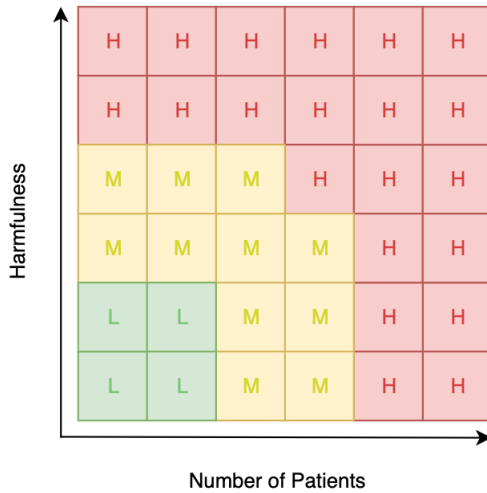


Figure 5.2: Impact matrix

Table 5.2: Description of STRIDE threats against the pacemaker device and their impact

| ID | STRIDE | Description | Impact | P/A |
|----|------------------------|--|--------|-----|
| 1 | Information Disclosure | An attacker connects to the pacemaker unauthorized and collects data from the pacemaker | Low | A |
| 2 | Tampering | An attacker connects to the pacemaker and tries to make reconfigurations | Medium | A |
| 3 | DoS | An attacker can send a signal to the pacemaker to ask for data continuously, which leads to the battery of the pacemaker draining rapidly | Medium | A |
| 4 | DoS | An attacker can jam the pacemaker with signals that the patient's heart is not beating as it should, which can lead to several unnecessary electrical pulses | Medium | A |

which had threats in the categories of information disclosure, tampering, and DoS. The reason why the threat with ID 1 has a low impact is that an attacker connecting to a pacemaker and collecting data can potentially harm that one individual patient. If a low-skilled malicious attacker is able to obtain patient data and has the goal of harming the patient, the attacker requires additional medical knowledge to comprehend the information. On the other hand, the impact increases on ID 2 as the attacker can do configurations that result in wrong and unsuitable treatment from both the pacemaker and healthcare personnel. This type of reasoning was practiced throughout to complete the tables of each component of the ecosystem.

The pacemaker app and the mobile phone's threats are handled in Table 5.3. This table includes the highest number of threats. It considers all vulnerabilities regarding both the application downloaded from Google Play Store and also weaknesses potentially found on the mobile phone itself. In addition, this part of the system is our focus area, and thus we studied these components in more detail. In contrast to threats in other parts of the ecosystem, these threats also depend on the behavior of the patient. For instance, the patient updating the phone and application to patch security gaps affects the level of security. If the patient lacks security knowledge or is unaware of having an outdated version of the software, the software can become vulnerable, which again can pose a risk to the safety and privacy of the patient.

Table 5.3: Description of STRIDE threats against the pacemaker app and the mobile phone and their impact

| ID | STRIDE | Description | Impact | P/A |
|----|------------------------|--|--------|-----|
| 5 | Spoofing | An attacker can pretend to be an authorized application that the patient downloads, then the attacker obtains health data from the device | Low | A |
| 6 | Spoofing | An attacker can bypass authentication with physical access to the mobile device and gain the view of a logged-in user in the app. This gives insight into the health data of the patient | Low | A |
| 7 | Tampering | An attacker can hijack the application and send wrongful data to the doctor or show false data to the patient, leading them to believe something is wrong | Medium | A |
| 8 | Spoofing and EoP | An attacker can exploit a known vulnerability in the OS of the patient's phone because the user has not updated their phone | Low | A |
| 9 | EoP | An attacker can brute force the password of the user with physical access to the mobile device, and is thus able to gain authorized access to the application | Low | A |
| 10 | Information Disclosure | The patient loses their login credentials, or shares them with someone else | Low | - |
| 11 | EoP | The patient's phone can be stolen, which makes it possible to gain physical access to the phone | Low | A |
| 12 | Spoofing and EoP | The attacker can exploit a bug resulting from a missing update of the application from the patient | Low | P |
| 13 | EoP | The attacker can gain access to other parts of the patient's mobile phone after hijacking the application, because of app permissions | Low | P |
| 14 | Information Disclosure | The attacker can use the app's login site to perform an SQL injection and gain access to data from one or multiple patients | Medium | A |
| 15 | Spoofing and Tampering | The attacker can gain access to the application and send instructions to the pacemaker to cause harm to the patient | Medium | A |

Table 5.4: Description of STRIDE threats against the data server and their impact

| ID | STRIDE | Description | Impact | P/A |
|----|------------------------|--|--------|-----|
| 16 | Tampering | An attacker can forge packets and send them to the server to corrupt the real data and trick the doctor into thinking something is wrong, potentially resulting in incorrect treatment | High | A |
| 17 | Information Disclosure | The patient data is not securely stored, and it is thus vulnerable to theft or misuse | High | - |
| 18 | DoS | The attacker repeatedly sends requests to the data server, making the server unavailable to access for healthcare personnel | High | A |

The data server has a few threats addressed in Table 5.4, which cover how patient data is handled and stored. The most common attacks against data storage were tampering, DoS attacks, and disclosure of information. This is because the data server is responsible for data at rest. All these threats could affect a large number of patients if they are realized, and therefore the level of impact is either high or medium for these threats.

When it comes to threats regarding the doctor’s tablet in Table 5.5, there is a potential to affect more than one singular patient if an attacker is able to connect, authenticate, or log in to one of the personnel’s tablets/computers. As a consequence, the impact is either medium or high for all the threats against this component. The threats classified as having a medium impact are the ones where there is no direct threat against the patients. On the other hand, for those classified as a high impact, the threat directly causes harm to the patients. We can observe that the doctor’s tablet has threats within several different categories of STRIDE.

Table 5.5: Description of STRIDE threats against the doctor's device and their impact

| ID | STRIDE | Description | Impact | P/A |
|----|------------------------|--|--------|-----|
| 19 | EoP | An attacker finds a way to gain unauthorized access to the system by stealing login credentials from authorized personnel | High | A |
| 20 | Spoofing | The attacker can bypass authentication on the doctor's computer application, and gain insight into several patients' data | High | A |
| 21 | Information Disclosure | The doctor/nurse loses their login credentials | Medium | - |
| 22 | Spoofing | Employees or other insiders who have access to the ecosystem could cause harm by misusing patient data or manipulating device settings | High | A |
| 23 | Repudiation | An attacker is logged into the system, pretending to be a medical professional, and later denies having done this malicious event | Medium | A |

Table 5.6 covers the data flow over the Wi-Fi network from the patient's mobile phone to where the patient data is stored and handled, and further to the doctor's tablet. The threats describe attacks concerning intercepting, blocking, and jamming signals, in addition to sniffing packets. We observe that the threats are ranging from medium to high, depending on the consequences they have for the patients. The other part of the data flow of the pacemaker ecosystem is in Table 5.7, which is the communication link between the pacemaker and the patient's mobile phone. The attacker can limit the availability of the service between the phone and pacemaker, as well as intercept the communication. In addition, the attacker can exploit vulnerabilities of BLE, which is the basis for the communication protocol between the pacemaker and phone. Both the data flow tables include attacks concerning tampering, information disclosure, and DoS only since the analysis focuses on the data being transmitted, not how the data is interpreted [14].

Table 5.6: Description of STRIDE threats against the communication over the Internet and their impact

| ID | STRIDE | Description | Impact | P/A |
|----|------------------------|--|--------|-----|
| 24 | DoS | An attacker can jam the signal from the doctor to the data server | Medium | A |
| 25 | Information Disclosure | An attacker can intercept the data on the interface between HMU/phone and the server | Medium | P |
| 26 | Tampering | An attacker can gain access to the communication flowing between the two devices, and alter the data during transit | High | A |
| 27 | Information Disclosure | The phone interacting with the pacemaker connects to an unsecured Wi-Fi network, making sensitive data vulnerable to interception/theft | Medium | P |
| 28 | DoS | An attacker can block or interrupt the transmission of the data going from the HMU/mobile phone to the data storage, so the doctor is unaware of the status of the patient | Medium | A |
| 29 | Tampering | An attacker can do a replay attack of the packets, and sniff health data on the communication link by getting unauthorized access | Medium | P |

To summarize, all identified threats against each individual DFD object in Medtronic's pacemaker ecosystem is represented in Table 5.8. The table displays a mapping where a specific DFD element is associated with one of the categories of STRIDE, where each identified threat is stated in the table with an ID.

Table 5.7: Description of STRIDE threats against the communication between the pacemaker and phone and their impact in the Medtronic ecosystem

| ID | STRIDE | Description | Impact | P/A |
|----|------------------------|---|--------|-----|
| 30 | DoS | An attacker can jam the signal from the pacemaker device to the mobile phone | Low | A |
| 31 | Tampering | An attacker can alter the traffic going from the pacemaker to the mobile phone, which can make the data sent to the doctor incorrect | High | A |
| 32 | Information Disclosure | An attacker can intercept the data on its way from the pacemaker to the phone | Low | P |
| 33 | DoS | An attacker can block the communication link between the phone and the pacemaker which prevents legitimate activity and results in patient data information being unavailable for a time period | Medium | A |
| 34 | Information Disclosure | An attacker can exploit known vulnerabilities in BLE, which could allow attackers to access sensitive patient data | Low | P&A |

Table 5.8: Summary of STRIDE categories with identified threats against the pacemaker's components in the Medtronic ecosystem

| Component | Spoof | Tamper | Repudiate | Info Disc | DoS | EoP |
|------------------------------|-----------------|--------|-----------|-----------|--------|------------------|
| Pacemaker device | | 1 | | 2 | 3, 4 | |
| App and Mobile Phone | 5, 6, 8, 12, 15 | 7, 15 | | 10, 14 | | 8, 9, 11, 12, 13 |
| Data server | | 16 | | 17 | 18 | |
| Doctor's tablet | 20, 22 | | 23 | 21 | | 19 |
| Dataflow: Cell/Wi-Fi Network | | 26, 29 | | 25, 27 | 24, 28 | |
| Dataflow: Bluetooth | | 31 | | 32, 34 | 30, 33 | |

5.1.3 What Are We Going to Do About It?

For question 3, we looked at what kind of measures Medtronic could do to manage the threats found in the threat modeling. Referring to subsection 4.1.3, there are four different ways to deal with the risk associated with these threats - eliminating, accepting, mitigating, or transferring. As mentioned in the mitigation section, it was hard for us as third parties to determine which risks are acceptable for Medtronic. In addition, we had little knowledge of their ability to transfer the risk to other parties in the manufacturing chain. Lastly, it is often hard to eliminate risk completely while still keeping all functionality intact. Therefore, our proposed countermeasures in this section focus only on mitigating the risks. In this way, we made sure that the system could remain as it currently is, but that the security was heightened. We provide an overview of the mitigation techniques for each component divided into the categories in STRIDE. The threats presented in the lists are clickable if there is a need to refresh on their wording.

Mitigation Techniques for Threats Against the Pacemaker Device

The following threats with IDs 1, 2, 3, and 4 can be found in Table 5.2.

- Tampering (ID 2):
 - Use intrusion and prevention systems to monitor and block suspicious activities in real-time
 - Implement access control to the functions of the pacemaker to restrict unauthorized modifications
 - Make sure that the pacemaker has closed all unnecessary interfaces to limit the connectivity to the real world
- Information Disclosure (ID 1):
 - Make sure that data at rest is encrypted, to prevent unauthorized people from gaining any knowledge from accessing the data
 - Implement access control to the data, to make sure that only authorized people can read the data stored in the database
- DoS (ID 3, 4):
 - Limit the number of requests sent to the pacemaker for a given time period
 - Implement authentication and access controls to ensure that authorized personnel can access the pacemaker
 - Implement only one-way communication between the pacemaker and mobile device

- Implement shielding material in the pacemaker to protect it from electromagnetic interference. The shield material should be designed to block radio waves or microwave radiation. Typical materials for this are metals, carbons, ceramics, and cement [16]

Mitigation Techniques for Threats Against the Pacemaker App and the Mobile Phone

In the next section, IDs from Table 5.3 are presented.

- Spoofing (ID 5, 6, 8, 12, 15):
 - Require password-based authentication with strong password criteria. Make sure that the patient has a limited number of attempts to log into the application, to prevent an attacker from brute forcing access
 - Require biometric or two-factor authentication to ensure only authorized users can access
 - Use intrusion and prevention systems to monitor and block suspicious activities in real-time
 - Give the patient sufficient knowledge on where they can find the correct application for their device
 - Give the user reminders to update the application whenever there is a new update available and ask the user to update their mobile phone when the SDK version is below a certain threshold
 - Educate patients about the importance of regularly updating the devices and applications. Also, encourage the patients to enable automatic updates
 - Give the patient sufficient knowledge on where they can find the correct application for their device
 - Provide the patient with a step-by-step guide on how to download the application when they receive a new pacemaker where the mobile application is substituting the HMU
 - Implement access controls to restrict access to health data stored in the device, which makes it unavailable for unauthorized attackers even if they manage to obtain it
- Tampering (ID 7, 15):
 - Apply strict integrity checks to the data being transferred between the server and the patient
 - Implement stronger authentication measures to the application to prevent unauthorized access

- Implement end-to-end encryption of the data to prevent altering of the information during transit. This can prevent someone from eavesdropping on the communication
- Information Disclosure (ID 10, 14):
 - Teach the patients about the importance of keeping their login credentials secure
 - Educate patients on the importance of changing their login credentials immediately if they notice that they are missing
 - Teach patients about secure ways to store their login credentials
 - Make it mandatory for users to change their password after a specific amount of time, to ensure that the same password is not in circulation for too long
 - Make sure that there is proper input validation in place, to prevent SQL injections
 - Parameterize the SQL queries, to separate the input data from the SQL code. Then the user input cannot be interpreted as a query from the database
- EoP (ID 8, 9, 11, 12, 13):
 - Make sure to always remind the patient to update their application whenever a new version is released
 - Teach the patients about the risks of not updating their applications and phone
 - Apply strong rules for passwords, to make it harder for an attacker to brute force the password
 - Give information to patients about the importance of keeping their devices secure, with login mechanisms to access their device
 - Keep strict control over which permissions the app asks of the user, to limit the amount of access a person gets to the rest of the system
 - Educate the patients on app permissions to make the user aware of what the app has access to on their phone and that they deny permissions if they do not wish to share certain items

Mitigation Techniques for Threats Against the Data Server

Table 5.4 contain the threats 16, 17, and 18. Their mitigation techniques are presented here.

- Tampering (ID 16):
 - Apply strict integrity checks to the data being transferred between the server and the patient/doctor
 - Implement end-to-end encryption of the data to prevent altering of the information during transit. This can prevent someone from eavesdropping on the communication
 - Enforce server-side validation so that the server validates the authenticity and integrity of the incoming data
- Information Disclosure (ID 17):
 - Make sure that data at rest is encrypted, to prevent unauthorized people from gaining any knowledge from accessing the data
 - Implement access control to the data, to make sure that only authorized people can read the data stored in the database
 - Perform regular backup of patient data so that in the case of theft it could be possible to recover the data
- DoS (ID 18):
 - Limit the number of requests sent to the data server for a given time period
 - Prevent a single point of failure by distributing and balancing the load of the data server

Mitigation Techniques for Threats Against the Doctor's Device

Table 5.5 shows threats against the device belonging to the doctor and mitigation techniques are the following:

- Spoofing (ID 20):
 - Require biometric or two-factor authentication to ensure only authorized personnel can access
 - Use intrusion and prevention systems to monitor and block suspicious activities in real-time
- Repudiation (ID 23):
 - Implement access control to the system, based on roles. This gives personnel only access to relevant information for their role, which can mitigate the amount of damage an attacker can execute

- Introduce a system that logs all users' activity in the system, to recognize suspicious activity faster
 - Add digital signatures to the actions performed by healthcare personnel. Thus it is not possible to deny having executed a task
- Information Disclosure (ID 21):
- Teach the personnel about the importance of keeping their login credentials secure
 - Educate personnel on the importance of changing their login credentials immediately if they notice that they are missing
 - Teach personnel about secure ways to store their login credentials
 - Make it mandatory for users to change their password after a specific amount of time, to ensure that the same password is not in circulation for too long
- EoP (ID 19):
- Give information to personnel about the importance of keeping their devices secure, with login mechanisms to access their device
 - Teach personnel about routines when noticing that their device has been used by someone else

Mitigation Techniques For Threats Against Communication Over the Internet

Communication over the internet has several threats, as seen in Table 5.6, and we present the mitigation techniques for them.

- Tampering (ID 26, 29):
- Apply strict integrity checks to the data being transferred between the server and the patient/doctor
 - Implement end-to-end encryption of the data to prevent altering of the information during transit. This can prevent someone from eavesdropping on the communication
 - Add a Message Authentication Code (MAC) to the messages being sent over the communication link, to prevent replay attacks
 - Use timestamps on the messages to prevent replay attacks
 - Implement strong access control to prevent unauthorized access to the system

- Information Disclosure (ID 25, 27):
 - Make sure that all data is encrypted during communication, to prevent eavesdropping and protect confidentiality
- DoS (ID 24, 28):
 - Implement frequency hopping or spread spectrum techniques to spread the signal over a wider range of frequencies at the same time period
 - Implement redundant communication channels, such as multiple wireless, resulting in a backup communication path
 - Make sure that all data is encrypted during communication, to prevent tampering or interception

Mitigation Techniques For Threats Against Communication Between the Pacemaker and Phone

Table 5.7 is the final table with threats for Medtronic, and their mitigation techniques are:

- Tampering (ID 31):
 - Apply strict integrity checks to the data being transferred between the pacemaker and the phone
 - Implement end-to-end encryption of the data to prevent altering of the information during transit. This can prevent someone from eavesdropping on the communication
- Information Disclosure (ID 32, 34):
 - Make sure that all data is encrypted during communication, to prevent eavesdropping and protect confidentiality
 - Keep up to date on known vulnerabilities in BLE and take appropriate measures to mitigate the impact of these vulnerabilities
- DoS (ID 30, 33):
 - Implement shielding material in the pacemaker to protect it from electromagnetic interference. The shield material should be designed to block radio waves or microwave radiation. Typical materials for this are metals, carbons, ceramics, and cement [16]
 - Implement frequency hopping or spread spectrum techniques to spread the signal over a wider range of frequencies at the same time period
 - Implement redundant communication channels, such as multiple wireless, resulting in a backup communication path

5.2 Biotronik's Ecosystem

We conducted a threat modeling of Biotronik's pacemaker system. The objective was to identify potential threats and vulnerabilities within the system and take measures to mitigate them in order to ensure the security of the patient. The initial step was to gain a comprehensive understanding of the architecture and design of the Biotronik system. Thereafter, we discussed the network components and analyzed potential threats that could occur. Finally, we proposed mitigation options as well.

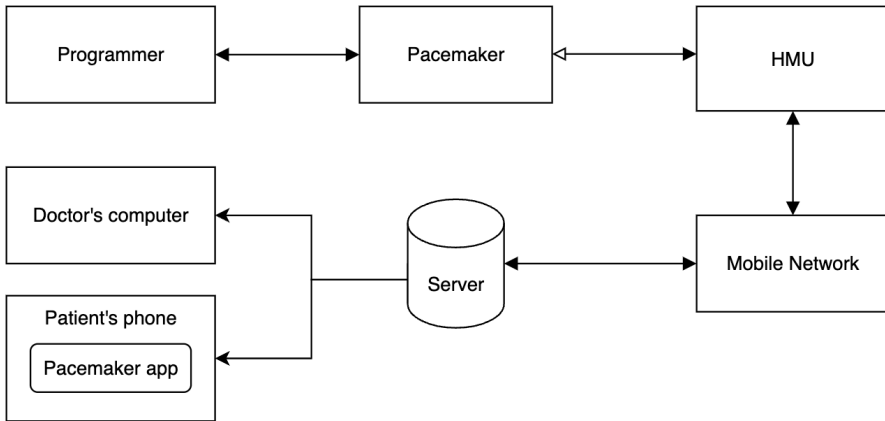
5.2.1 What Are We Working On?

The first step in the threat modeling of the Biotronik system was to get an understanding of the system model. The threat model included a full breakdown of processes, data stores, data flows, and trust boundaries to answer this section's title. This was used as a foundation to find out what can go wrong in Biotronik's system. The models in Figure 5.3 illustrate the data flow diagram both in the high- and mid-level of the vendor's pacemaker ecosystem. DFD was used to structure the system models. Additional elements used in Biotronik's pacemaker ecosystem are a Virtual Private Network (VPN). Also in this figure, the backend infrastructure is displayed as a black box, representing a system that can be viewed by its inputs and outputs. The black box is incorporated into the system model since we have limited knowledge and understanding of its internal workings. As for the VPN, we know that the communication between the HMU and the infrastructure is done securely over a VPN connection.

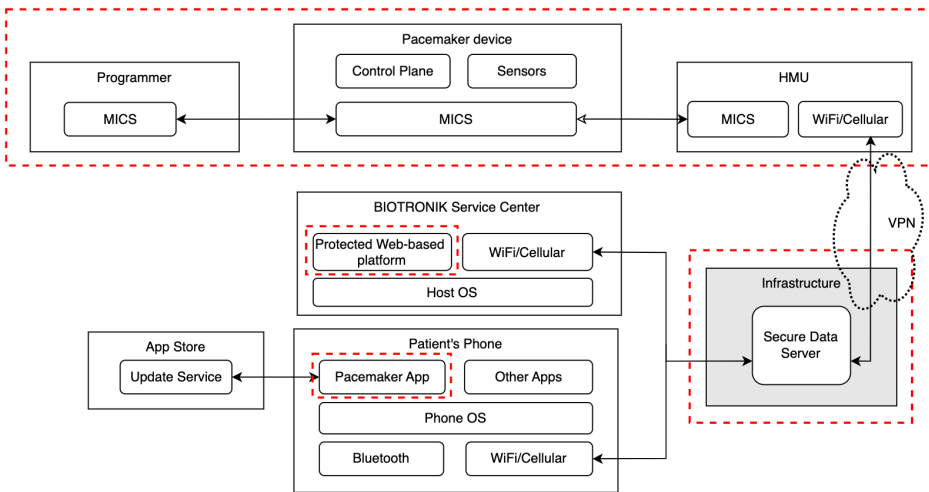
Table 5.9 includes a summary of the different components in the Biotronik ecosystem. We observed that there were some differences between the Medtronic system and this one. In particular, the Biotronik ecosystem used a different protocol, Medical Implant Communication System (MICS), for the communication between the pacemaker and the HMU. Additionally, communication over the internet between the HMU and the backend was behind a VPN connection.

5.2.2 What Can Go Wrong?

The STRIDE model was used in the Biotronik system as well. We constructed different tables for the different DFD objects of the ecosystem, the threats associated with the element, and the impact of these threats. To avoid unnecessary repetition, we refer to the previous tables from the Medtronic STRIDE analysis in subsection 5.1.2 when the threats appear to be identical. The threats uncovered were classified into one of the six STRIDE categories, and given an impact level of low, medium, or high. The definition of the categories and the impact levels used in subsection 5.1.2 remains the same. A summary of all the threats found is presented in Table 5.12.



(a) High-level data flow diagram



(b) Mid-level data flow diagram

Figure 5.3: Data flow diagram of the Biotronik's ecosystem

Table 5.9: Identified pacemaker components and their description in the Biotronik's ecosystem

| DFD Object | Description |
|----------------------------------|---|
| Pacemaker device | This is the device situated inside the patient's chest, responsible for giving small electrical pulses to keep the heart functioning as normal |
| App | This is an Android application on the patient's phone. The app is showing some statistics to the user, that it has received from the data server |
| Programmer | This is the device responsible for programming the pacemaker to fit each patient's specific needs. Resides at the hospital, and is operated by healthcare personnel |
| HMU | This is a small computer receiving medical information from a pacemaker. It is placed in the patient's home and it makes it possible for healthcare personnel to receive the data. This simplifies monitoring for the patient as she does not need to visit the doctor as often |
| Data server | The server of the vendor collects and exports the data coming from the HMU of the patient. The data is accessible to healthcare personnel, and a part of the data is accessible to the patient |
| Service Center | The computer/tablet belongs to the healthcare personnel. The tablet has an application with information about the different patients the doctor is responsible for. Can gather data from the data server |
| Dataflow: Cellular/Wi-Fi Network | The data flow goes from the HMU to the data server, containing all the data from the pacemaker. The doctor can gather data from the data server to their local computer/tablet over cellular/WiFi |
| Dataflow: MICS | The dataflow going from the pacemaker to the HMU and between the pacemaker and the programmer. A proprietary protocol made specifically for IMDs communication |

The two ecosystems are somewhat similar, thus some of the tables from Medtronic were directly equivalent for Biotronik. However, there were some differences. The tables not applicable to the Biotronik ecosystem were the one detailing communication between pacemaker and phone (Table 5.7), and parts of the table on mobile phone threats (Table 5.3). This was mainly a result of the ecosystem's use of different protocols for communication and the mobile application's lack of communication with the pacemaker. Therefore, two additional tables are included in this section, one for the HMU device itself, and another for the communication between HMU and the pacemaker. In addition, necessary adjustments were made to the other tables to reflect this version of the ecosystem.

There exist tables that can be directly used for the Biotronik system. The threats regarding the pacemaker device, presented in Table 5.2 are applicable as they are, and we are not making any adjustments to this system. Further, Table 5.4, and Table 5.5, the tables regarding threats to the servers in the backend infrastructure and the threats against the hospital personnel are directly applicable to the Biotronik system. It is worth noting that the device used by doctors and personnel at the hospital has different names in the two models. For Medtronic, they refer to it as the doctor's tablet, containing two different applications for communication with the programmer and the data server. In Biotronik's case, the device is called a Service Center, and this contains a web-based application for communication with the backend. Despite that these systems are named differently, their function in the ecosystem is the same and therefore have identical threats.

When it comes to communication over Wi-Fi, between the HMU, backend infrastructure, the hospital's devices, and the patient's phone, we can use Table 5.6 with some modifications. As mentioned earlier, the communication between the HMU and the infrastructure goes through a VPN. Thus, communication should be more secure than an open connection over the internet. Therefore, the threats with ID 25 (intercepting data on the interface) and ID 28 (blocking or interrupting the transmission) are not necessarily applicable to Biotronik's system. However, we are aware that the use of VPN does not make the communication completely secure against all attacks. This means the threats in question are becoming harder to execute, but not impossible. We include these in the threat model for Biotronik as well, even though the chances of these attacks succeeding are significantly lower than for Medtronic's case.

The table that describes the threats against the mobile phone and the application, Table 5.3, is most relevant for Biotronik as well. Since the application does not communicate directly with the pacemaker, some threats are simply not plausible for this system. In particular, ID 15 (sending instructions to the pacemaker) is not applicable to Biotronik. In addition, the threat with ID 7 (hijacking the application) is only partly relevant, as an attacker is not able to send data to the doctor directly. However, the attacker can show wrongful data to the patient, which can lead to them thinking something is wrong, or they can withhold information about an issue. Otherwise, the threats presented in Table 5.3 are also present in the Biotronik system.

As mentioned above, we made two new tables, one for the HMU itself (Table 5.10), and one for the communication between the pacemaker and the HMU (Table 5.11). These are the components that are not present in Medtronic's system and are thus unique threats to Biotronik.

Table 5.10: Description of STRIDE threats against the hardware HMU and their impact in Biotronik

| ID | STRIDE | Description | Impact | P/A |
|----|--------|---|--------|-----|
| 35 | DoS | An attacker can jam the HMU with signals in order to overwhelm the device, and prevent normal operation | Medium | A |
| 36 | EoP | An attacker can gain physical access to the device and obtain patient data | Low | A |

Table 5.11: Description of STRIDE threats against the communication between HMU and pacemaker and their impact in Biotronik

| ID | STRIDE | Description | Impact | P/A |
|----|-----------|---|--------|-----|
| 37 | DoS | An attacker can block the communication link between the HMU and the pacemaker, which prevents legitimate activity, resulting in patient data being unavailable | Medium | A |
| 38 | Tampering | An attacker can alter the packets going between the pacemaker and HMU, which can make data stored at the server untrue | Medium | A |
| 39 | DoS | An attacker can jam the signal from the pacemaker device to the HMU | Low | A |

As we observe in Table 5.10, there are two threats against the hardware HMU, and these have a low to medium impact. All the attacks against an HMU only affect one single patient, as it is connected to their personal pacemaker. In addition, the threats can affect the operation of the HMU, but this is not life-threatening for the patient. We also observe in Table 5.11 that the communication link between the pacemaker and the HMU is subject to threats ranging from low to medium. It can be quite harmful to the patient if the doctor does not receive the correct information about the patient's health, but it still affects only one patient at a time.

For the summary of all the threats applicable to Biotronik's system, we combined the threats taken from the tables in subsection 5.1.2 with the unique threats for Biotronik presented in this section, illustrated in Table 5.12.

Table 5.12: Summary of STRIDE categories with identified threats against the pacemaker's components in the Biotronik ecosystem

| Component | Spoof | Tamper | Repudiate | Info Disc | DoS | EoP |
|------------------------------|-------------|--------|-----------|-----------|--------|------------------|
| Pacemaker device | | 1 | | 2 | 3, 4 | |
| App and Mobile Phone | 5, 6, 8, 12 | 7, 15 | | 10, 14 | | 8, 9, 11, 12, 13 |
| HMU | | | | | 35 | 36 |
| Data server | | 16 | | 17 | 18 | |
| Service Center | 20, 22 | | 23 | 21 | | 19 |
| Dataflow: Cell/Wi-Fi Network | | 26, 29 | | 25, 27 | 24, 28 | |
| Dataflow: MICS | | 38 | | | 37, 39 | |

5.2.3 What Are We Going to Do About It?

We answered question 3 by finding mitigation techniques for the risks associated with the threats found. As a lot of the threats for the system are the same as the ones for Medtronic's system, we went through solely the ones that are unique for Biotronik for this section. For the common threats, we refer to subsection 5.1.3. We divided the threats into components where the list has categories based on the STRIDE model.

Mitigation Techniques for Threats Against the Hardware HMU

The next section presents the countermeasures for the threat with ID 35 and 36 from Table 5.11.

- DoS (ID 35):
 - Implement shielding material in the design of the HMU to protect it from electromagnetic interference
 - Implement frequency hopping or spread spectrum techniques to spread the signal over a wider range of frequencies at the same time period
 - Implement monitoring to detect signal interference and alert personnel to take proper action

- Use signal filtering to filter out unwanted signals and prevent the HMU from becoming overwhelmed
- EoP (ID 36):
 - Implement encryption techniques to protect the data stored on the device. Makes it unavailable for unauthorized attackers even if they manage to obtain it
 - Ensure that the HMU is regularly updated with up-to-date software and firmware to fix known vulnerabilities
 - Implement access control to the data on the HMU to ensure that only authorized people will have access to the information

Mitigation Techniques for Threats Against the Communication Between the HMU and Pacemaker

For the final techniques, we present threats 37, 38, and 39, revisit ?? for the threats.

- Tampering (ID 38):
 - Make sure that the communication going between the HMU and pacemaker is encrypted securely, to prevent eavesdropping
 - Implement integrity checks on the communication, to prevent alterations from going unnoticed
 - Add server-side validation, to make sure that the server validates the authenticity of the communicating parties
- DoS (ID 37, 39):
 - Implement shielding material in the design of the HMU to protect it from electromagnetic interference
 - Implement frequency hopping or spread spectrum techniques to spread the signal over a wider range of frequencies at the same time period
 - Implement monitoring to detect signal interference and alert personnel to take proper action
 - Use signal filtering to filter out unwanted signals and prevent the HMU from becoming overwhelmed
 - Implement redundant communication channels, such as multiple wireless, resulting in a backup communication path
 - Use intrusion and prevention systems to monitor and block suspicious activities in real-time
 - Make sure that all data is encrypted during communication, to prevent tampering or intercept

Chapter 6

Security Analysis of Applications

We research the mobile applications launched by Medtronic and Biotronik. The applications have the important goal of improving the quality of communication between a patient and a doctor. For the security analysis, we utilize several different tools for static analysis and a reverse engineering process. As mentioned in the analysis of the application section of our methodology, subsection 4.3.3, the tools are used to assess the security level of the apps and search for vulnerabilities. The overall goal of this security analysis is to gain insight into what vulnerabilities might be present in the applications, and how these can be exploited by an attacker. We try to gain access to the applications and see if we are able to access information that was not meant for us. This analysis is the foundation for answering our research question, and it helped us execute research objective number 1, “*Analyzing the mobile healthcare applications to see if they have an adequate level of security in regard to patient data and privacy.*”

Firstly, MobSF gave the foundation of the static analysis. In addition, we used an online tool called BeVigil for another perspective on static analysis. The reason for this was that we wanted to make sure that we were not bound by the results given from one tool, and thus stuck in these findings. By using two tools we were able to broaden the range of results to further investigate and increase overall effectiveness. Additionally, this could be used to detect potential false positives. A false positive is when one of the tools flags a part of the code as a potential vulnerability or issue, but in reality, it is not. By using two different tools, we compared and cross-validated the potential vulnerabilities, and used this to highlight genuine problems. For the static analysis, we used the results from MobSF as the basis and then supplemented them with information from BeVigil to give a more comprehensive view of the existing vulnerabilities.

After the results from the static analysis had been presented, we looked into the code more actively, by using different tools for reverse engineering the application. Our overall goal was to gain as much understanding of the application as possible.

Thus, we explored whether the vulnerabilities found in the static analysis were real issues for the application and checked if the existing vulnerabilities could be exploited. Two important tools for the more dynamic part of the analysis were mitmproxy and Burp Suite. Both of these were used as HTTP proxies that connected to our Android Emulator, in order to intercept the traffic going between the device and the backend servers. We used both of these tools as they had some different attributes. Burp Suite has a more interactive interface, which made it easy for us to understand how the communication is connected (appendix A.6). On the other hand, mitmproxy is useful for making fake responses to the requests coming from the device much easier and sending these automatically. This was because we could attach custom Python scripts to mitmproxy while it was running (appendix A.7). The usage of these two tools is explained further in the upcoming sections.

6.1 MyCareLink Heart App - Medtronic

MyCareLink Heart is a mobile application for pacemaker patients with a device supporting BlueSync technology. The purpose of the application is to enable patients with a Medtronic heart device with Bluetooth to download and use the application for remote monitoring. The app can send health data automatically and directly to the doctor. It also allows patients to conveniently access the data from their own implanted devices. The application requires the availability of a mobile phone or tablet, Bluetooth connectivity, and Wi-Fi to pull data from their pacemakers [51].

For the analysis of the MyCareLink Heart application, we began by presenting the results from the static analysis, performed by MobSF and BeVigil, as presented in section 4.3.3. Afterward, we looked further into the permissions that the application requested from the user. We performed an analysis of the APK for the application, where we looked at the code base and other files in the APK. There is a brief explanation of the structure of the application, found by analyzing the code base. Lastly, we provide a summary of the most important findings from the analysis.

6.1.1 Static Analysis

To start, we performed a static analysis of the mobile application. The result of the security assessment executed by MobSF gave an overall score of 41 points out of 100 for Medtronic's healthcare application. On the other hand, the BeVigil analysis gave a score of 8.7 out of 10, i.e. 87 out of 100 [35]. We could observe that these two tools presented significantly different results for the analysis, and a further look into what the two analyses focused on was necessary. The categorization of risk given by MobSF, shown in section 4.3.3, indicated that the application was on the verge of belonging to the high-risk category with an upper limit of 40 points, but was

assigned to medium risk. Following the same categorization, the BeVigil analysis gave the application a low risk.

The two analyses had quite different categories on which they based their score, even though the content was quite similar. The BeVigil analysis had a few extra categories, and thus we added the extra information from these at the end. We used the categorization from MobSF as a base and added the corresponding results from the BeVigil analysis. All the results for the BeVigil analysis were collected from [35].

Application Permissions

The two analyses did not agree on the level of severity of some of the permissions used in the application. They had both marked the location permissions as non-safe, with the MobSF analysis marking these as *Dangerous*, and BeVigil saying they were *Risky*, the level below *Dangerous*. In addition, the BeVigil included another permission that they considered troublesome, which has the level *Dangerous*. We look further into these:

- android.permission.ACCESS_BACKGROUND_LOCATION allows the application to access location in the background,
- android.permission.ACCESS_COARSE_LOCATION allows the application to access coarse location sources, the approximate phone location
- android.permission.ACCESS_FINE_LOCATION allows the application to access fine location sources, giving the most precise location
- android.permission.BATTERY_STATS allows an application to collect battery statistics (only in BeVigil analysis)

The majority of these permissions concern the application’s ability to access the location, either approximately or precisely, at any time. These can have privacy consequences in that a malicious app can track the patient’s location. To access coarse location, the mobile network database can be utilized to receive the approximate location. For the fine location, the available location providers, including GPS, Wi-Fi, and mobile cell data, obtain the precise location of the device [47]. Furthermore, the third-mentioned permission regarding fine location requires more capacity, naturally resulting in additional battery consumption.

When looking through the documentation for Bluetooth permissions on the Android Developer pages, [74], it was clear that some of these location permissions were associated with the use of Bluetooth as the communication medium. In particular, there was a need for location permissions when using Bluetooth and

supporting older Android versions. There are different needs for location permission based on the version of Android OS. Therefore, as this application is intended to support multiple Android versions, there is a necessity to ask permission for all of these location services. This is because it would be difficult to request different permissions based on which version of the Android OS a specific phone supports, as the permissions must be stated in the manifest file. We discuss further how these permissions are used in the upcoming sections.

When it comes to the battery statistic permission, this allows the application to collect different statistics about the phone's battery. This includes reading the current low-level battery use data. The issue with this permission is that it may also allow the application to find out detailed information about which apps are used the most on the device [3]. The reason is that the low-level battery data can include how much of the battery is consumed by the different applications on the phone. By allowing this permission, the user is to some extent allowing the application to gain access to data of their phone usage.

Users must grant dangerous permissions to the application when it is running. Therefore, the above-mentioned permissions are *Runtime* permissions, which means that the user has to take explicit action to approve the permissions. Upon the first use of the application, the user is prompted with a question to allow the application to access these parts of your system, often with the option to “allow while using the app”. The permissions are retained, but the users have the possibility to change them [67]. Android applications need to request these permissions since security measures are implemented by Android OS to protect the user's privacy and their sensitive data. Runtime permissions give the user a choice to access or deny them and thus have more control over the information they share. This ensures that the user is aware of what the application can access and grant. If the user wishes to deny, the applications might not work as intended, or some functionality might not be accessible anymore.

Certificate Analysis

One category that was only present in MobSF was the certificate analysis. This is an analysis of the certificates associated with the application, and there were two different warnings presented. The first one was that the application was vulnerable to Janus Vulnerability, as it was signed with a v1 signature scheme. In other terms, the signature algorithm was SHA-1 with RSA, a weak algorithm that is explained in the next paragraph. This is a vulnerability that makes it possible for an attacker to modify an application undetected. This is done by adding a malicious Dalvik executable (DEX) file to an APK file. This addition goes unnoticed because when the signature is checked for alterations, only the APK file is detected, not the malicious

DEX file [26]. If the application runs on Android 5.0-7.0 it is vulnerable to this whether it is signed with a v1, v2, or v3 scheme. If the application is signed with a v2 or v3 signature scheme, then the signature algorithm is SHA-256 with RSA or SHA512 with RSA, respectively. For applications running on Android 7.0-8.0 the vulnerability is present only if the signature scheme used is v1.

The other vulnerability presented in the application was that the certificate algorithm might be vulnerable to a hash collision. The reason for this was that the application was signed with SHA-1 with RSA, as mentioned above. SHA-1 algorithms are known to have collision issues, which means that it is possible for an attacker to create a fake certificate with the same hash value as the original one. This can lead to potential security vulnerabilities, such as impersonation and data tampering. This hash function was theoretically broken back in 2005, and the first successful collision attack occurred in 2017. In 2019 there was proof of a chosen-prefix collision, making it possible for the attacker to choose the prefix for two colliding messages [17]. Thus, it is possible to have collisions with important data inside, not just arbitrary collisions. In practice, this makes SHA-1 very vulnerable to collision attacks. However, MobSF stated that the manifest file suggested that the application used SHA-256 with RSA. This indicated that the app used a different certificate for signing. This was a more secure algorithm, which could mean that the overall security impact on the app was not that severe. Further analysis of the certificate algorithm is performed later, see section 6.1.3.

Manifest Analysis

For the analysis of the manifest file associated with the application, the two analyses reported different findings. The MobSF analysis found some issues, while BeVigil did not report any findings. For MobSF there was one finding with a severity level of high, and some warnings. The issue with the severity high was that clear text traffic is enabled for the application. This can be seen in the *android:usesCleartextTraffic=True* in the manifest.xml file. The app could use cleartext network traffic, such as cleartext HTTP and FTP. This issue was of high severity as cleartext traffic meant that it could be possible for an attacker to eavesdrop on the communication and also modify it without being detected. This is a clear violation of confidentiality and authenticity, and it does not protect against tampering. The default value for this field for apps that target API level 27 or lower is “true”, while it is “false” for those targeting API level 28 or higher [1].

In addition to this issue, there were several warnings present in the manifest analysis. These are issues that could be a vulnerability, depending on the implementation. The first warning was that the application could be installed on a vulnerable Android version. If the app was installed on an older version of Android with unfixed

vulnerabilities, this also made the application itself more exposed to threats. The application was likely made to have opportunities to support several Android versions, as it was in use by a large patient group with several different cell phones.

The next warning was that a broadcast receiver in the application was protected by a permission, but the protection level of the permission should be checked. A broadcast receiver is a component in the Android system that makes the device receive and respond to system-wide broadcast announcements, such as low-battery notifications or incoming calls. This issue could be a problem because the broadcast receiver is accessible to any other application on the phone, as these receivers are shared with other apps. If the permission was not strict enough, a malicious application could request and obtain permission and interact with the component. The reason for this only being a warning was that the permission was not defined in the analyzed application, and thus MobSF did not know the level of the permission.

The last warning of the manifest analysis was that `TaskAffinity` was set for activity. `TaskAffinity` is an attribute that is defined in each `<activity>` in the Manifest. It describes which task an activity prefers to join. Activities in the same task share the same back stack, which allows the user to navigate between them using the back button. The default is to set the task as the package name, to prevent information from being leaked. If this is not done, other applications could read the intents, which are the actions to be performed, that are sent to activities belonging to another task. This means that if there is any sensitive information, such as user data, being sent with the intent, this can be read by another application. In addition, there is an attack known as Task Hijacking that can be performed on the application's back stack. This is done by making a malicious application that is automatically pushed to the back stack of a vulnerable app, by setting the task of the application's activity to the same as the task of the vulnerable app. Whenever the user tries to open the vulnerable app, the malicious app is present in the back stack, and therefore the user is met by the malicious app's activity when he presses the back button [38].

Code Analysis

For the code analysis of the MyCareLink Heart App by MobSF, threats with severity categorized as info and warning are covered. To achieve a clear overview of the obtained outcome of the code analysis, see Table 6.1. It was divided into four columns; the threat ID, the severity, information about the issue, and the related standards discussed in section 3.2. The table highlights possible vulnerabilities within a static source code. Each potential vulnerability is connected to one or more standards informing about a security verification requirement or a security breach.

There were three issues from the Table 6.1 that have the severity of Info. This means that they are not necessarily a threat to the system, and thus we explain

Table 6.1: The result of the code analysis by MobSF of MyCareLink Heart App and Biotronik

| ID | Severity | Info | Standards |
|----|----------|---|--|
| 1 | Info | The app logs information. Sensitive information should never be logged. | CWE: CWE-532: Insertion of Sensitive Information into Log File OWASP MASVS: MSTG-STORAGE-3 |
| 2 | Info | This app uses SQL Cipher. Ensure that secrets are not hardcoded in code | OWASP MASVS: MSTG-CRYPTO-1 |
| 3 | Warning | App uses SQLite Database and executes raw SQL query. Untrusted user input in raw SQL queries can cause SQL Injection. Also, sensitive data should be encrypted and written to the database. | CWE: CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') OWASP Top 10: M7: Client Code Quality |
| 4 | Info | This app listens to Clipboard changes. Some malware also listens to Clipboard changes. | OWASP MASVS: MSTG-PLATFORM-4 |
| 5 | Warning | The app uses an insecure Random Number Generator | CWE: CWE-330: Use of Insufficiently Random Values OWASP Top 10: M5: Insufficient Cryptography OWASP MASVS: MSTG-CRYPTO-6 |
| 6 | Warning | Files may contain hardcoded sensitive information like usernames, passwords, keys, etc | CWE: CWE-312: Cleartext Storage of Sensitive Information OWASP Top 10: M9: Reverse Engineering OWASP MASVS: MSTG-STORAGE 14 |

them briefly. ID 1 is regarding the fact that the application logs information, with a reminder that sensitive information should never be logged. If sensitive data is logged, this is a violation of CWE-532 regarding the opportunity to expose sensitive user information through information written to log files. This problem is also addressed in OWASP MASVS. ID 2 gives details about the cipher used in the application, which is SQLCipher. This extends the normal SQLite database library to improve security and facilitate encrypted local storage [71]. Here as well it is recommended

to make sure that there are no hardcoded secrets in the code. This kind of issue is only mentioned in the OWASP MASVS. The last issue with severity Info is ID 4, related to the app listening to clipboard changes. This issue is mostly related to the fact that malicious applications also could listen to these changes. This means that if a user were to copy something from the application, this can be read by another application on the device [7]. This issue is also mentioned in the OWASP MASVS.

ID 3 in the Table 6.1 is the first issue with a severity Warning. This issue is related to the use of SQLite databases, with the execution of raw SQL queries. Because of the raw query, there can be an opportunity for SQL injection from untrusted user inputs. This issue is mentioned in CWE-89, which is about SQL injections. The CWE definition is that a product constructs a SQL command with an externally-influenced input, but does not execute proper sanitization of special elements, before sending this to another component [21]. This issue is also represented on the Top 10 from OWASP, under M7: Client Code Quality. This risk relates to the fact that bad code quality can introduce opportunities for an attacker to exploit weaknesses in the code, without needing many skills.

Referring to Table 6.1, we take a closer look at the application's issue with ID 5. It reports an insecure use of the Random Number Generator. The issue belongs under CWE-330 regarding the insufficient use of random numbers and values in a security context that relies on unpredictable numbers. Besides, the issue relates to one of the OWASP's Top 10 Mobile Risks from 2016, in fact, M5 about insufficient cryptography. The applications may pose the risk of improper encryption if the Random Number Generator is insecure. Additionally, the threat falls within the third standard OWASP MASVS. This security requirement concerns data storage and privacy by encrypting sensitive data and use of authentication. The applications may not meet this requirement. The standards reveal the importance of the vulnerability's potential impact and the need to address it.

The last issue in the list, ID 6, also has a severity of Warning. This issue is related to the fact that files might contain hardcoded sensitive information, such as usernames and passwords. This issue told us that we might find hardcoded secrets if we look for these in the application source code, as there were several hardcoded strings available. This issue is related to CWE-312, regarding cleartext storage of sensitive information within a resource that can be accessible to another controller. This issue is also listed in the OWASP Top 10, under M9: Reverse Engineering. This is because hardcoded strings are one of the first things one looks for during a reverse engineering process. Thus, if sensitive information is stored in a hardcoded way, this can even be found by attackers with low skills. This particular issue is also found in OWASP MASVS under the storage category, stating that if sensitive data is required to be stored locally, this information should be encrypted with a key stored

in hardware that requires authentication to access.

Table 6.2: The result of the code analysis by BeVigil of MyCareLink Heart App

| ID | Severity | Info | Standards |
|----|----------|---|--|
| 7 | Medium | The app uses Non-Parameterized SQL queries, which makes the application vulnerable to SQL injection where the attacker can inject malicious SQL statements to exfiltrate the data from the database | CWE: CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') |
| 8 | Medium | The app uses implicit intent for broadcast. Applications that send broadcasts without specifying the target may have these intercepted by malicious apps on the same device. | CWE: CWE-927: Use of Implicit Intent for Sensitive Communication |
| 9 | Medium | Storage of sensitive information in shared preference. Anyone with root access to the device can be able to read the information stored in shared preference, and can thus compromise this information. | CWE: CWE-312: Cleartext Storage of Sensitive Information |

The BeVigil analysis highlighted some other vulnerabilities in the code. We collected the findings from the BeVigil analysis in Table 6.2. All the issues presented have the severity “Medium”. Some of the issues are related to things that were brought up in Table 6.1, for example, we can see that ID 3 is related to ID 7, and ID 6 is related to ID 9. These are also connected to the same Common Weakness Enumeration (CWE) definitions.

The vulnerability with ID 7 is regarding the SQL database in use in the application. An extra feature of the BeVigil analysis which is not present in the MobSF is the possibility to see which files the issues are coming from. This particular issue, with the non-parameterized SQL queries, is related to a file within the Google folder of the application. Specifically, it is a *SQLiteEventStore* file, within a *datatransport/runtime/scheduling* folder. Within this file, there is a *rawQuery()*, that sends a non-parameterized SQL query. To parameterize a query means to place a parameter in the query instead of a constant value. This allows the query to be reused with different values for different purposes. When the fields do not get parameterized, the user can insert a string directly into the query. This makes it possible to add SQL

phrases into the field, and thus the user can try to collect more information from the database, e.g. by asking the query to dump all the content of the table. When the field uses parameters, the user input is added to a pre-prepared query, which makes it harder to insert SQL expressions.

The second issue discussed in the analysis regards the use of implicit broadcasts. If the broadcast does not specify a target, it is possible for all the other applications on the device to listen to the broadcast. This means that if a user has an application on their device that is malicious, the app can gain access to the information being sent over the broadcast from other applications. Any application can register to an implicit broadcast by using an `IntentFilter` and declaring it in the manifest. This issue was connected to 4 different files, two of which were situated within the Google folder, and two within the Urbanairship¹ folder. In all these files there is an `Intent((...)).sendBroadcast(x)` method.

The last issue in the BeVigil analysis has ID 9 and is related to the storage of sensitive information in shared preference. This issue is only connected to one file in the application, situated within the Firebase folder. The file in question is called *Store.java* and it includes a `getSharedPreferences()` method. This can become a vulnerability, as shared preference is a storage that is shared with other applications on the device. If an attacker gains root access to the device, he can be able to read everything that is stored in shared preference. In this particular case, the method seems to be putting a token key in the storage, which would be possible to read outside of the application.

Shared Library Binary Analysis

For the shared library analysis, MobSF went through all the libraries used in the application and checked if these have any issues associated with them. For the Medtronic application, there was only one issue associated with shared library binaries in the MobSF analysis, while BeVigil did not report on any issues. The issue found by the analysis had high severity. The issue was connected to stack canaries, where the library *lib/x86_64/libxamarin-app.so* did not have a stack canary value added to the stack. Stack canaries are used to detect and prevent a stack buffer overflow before malicious code can replace return addresses. If an incorrect canary is detected during critical parts of the flow, the application will be terminated when stack canaries are present [73]. It could pose a significant issue since the library did not have a canary value. Therefore, it is recommended to enable stack canaries for best security practices. Although a stack canary was used as a mitigation technique, the absence of a stack canary did not automatically make the mobile application

¹UrbanAirship is a US-based company offering marketing and branding

```

<string name="google_api_key">AIza[REDACTED]</string>
<string name="google_app_id">1:83[REDACTED]</string>
<string name="google_crash_reporting_api_key">AIza[REDACTED]</string>

```

Figure 6.1: The Google API key and Google Crash Reporting API key in strings.xml

vulnerable. However, it is important to notice that the absence increases the risk if there exists a stack-based buffer overflow vulnerability.

Additional Information from BeVigil

The analysis done by BeVigil had a larger amount of categories being checked for vulnerabilities. We briefly run through the findings that have not been discussed previously. First, the analysis found an issue with severity medium in the analysis of strings, which was “Possible Secret Detected” in three different files, similar to the warning 6 in Table 6.1. All of these were found within the same file, which was a *strings.xml* file within the resources folder.

Two of the three strings in question are API keys for Google; `google_api_key` and `google_crash_reporting_api_key`, see Figure 6.1. The first-mentioned is used for the authentication and authorization of several Google APIs. Further, the second API key associated with crash reporting is integrating Google Crash Reporting into the mobile application. It is used to authenticate and authorize access to the Crash Reporting API. On Google’s own documentation site, it says that it is not necessary to hide your Firebase API keys [40]. These values are identifiers for Firebase and Google services that the client needs to know to access those services. However, Google provides general guidance on API security best practices. Here, it recommends taking action to protect your project against API key misuse [4]. The last string is a `password_toggle_content_description` that says “Show password”. This did not seem to have a large security impact.

BeVigil also analyzed the associated assets to the application and found an issue connected to the Firebase URL. This had a severity medium, and was connected to CWE-200: “Exposure of Sensitive Information to an Unauthorized Actor”. The URL for the cloud service was detected. It is important to ensure that this cloud service has proper authentication and authorization mechanisms in place. When visiting this URL address, it was not possible for us to access the contents, so it seems as though there are some access restrictions in place.

In addition, the analysis gave some information about what kind of trackers are present in the application. This gave insight into which companies gained access to analytical data and where the crash reports were sent. For the Medtronic application

both Google, Microsoft, and Urbanairship were used for analytics, while Microsoft and Google Firebase received the crash reports. The crash reporting key for Google is already discussed, and Microsoft relies on its App Center Crashes API for Android to receive crash information. Lastly, the analysis gave some general knowledge on the *classes.dex* file, such as whether it used an obfuscator, what kind of compiler it used, and if it contained any anti-VM measures. These findings are discussed in more detail in subsection 6.1.3.

6.1.2 Permissions

For the Medtronic application, there were several permissions that were marked as suspicious from the static analyses we performed. First of all, the application asked for all the different levels of location permissions, namely “fine”, “coarse”, and “background” location. In addition, the application asked to access the battery statistics of the device. In this section we dive a bit deeper into why the application asked for these permissions, and whether they could be an issue for the user.

We looked at the location permissions first. As stated in the Android Developer pages, an application must ask permission to access “fine” or “coarse” location, if they wish to run location services in the foreground [47]. This means that the application asks to use the location of the user for an activity that is present when the app is in active use by the patient. The developer only needs to ask for the “fine” location, if the application is in need of a precise location access. This level of location accuracy is usually within 50 meters and can be as little as within a few meters. For access to background location, the application needs to specifically ask for this, as long as it supports devices that use Android 10 (API level 29) and higher. Access to the location in the background means that as long as the application is running on the device, the app is able to collect the location of the user. The precision of the location is the same as for the foreground location, which means that in Medtronic’s case, the application keeps track of the precise location of the user, even if it is running in the background [47].

A potential reason for the need to access the user’s location is the use of Bluetooth as a communication method in the Medtronic application. When looking through the manifest file of the Medtronic application, we observed that within the permissions, the app asks to access “bluetooth” and “bluetooth_admin”. Both of these permissions are needed to request legacy Bluetooth on older devices, according to Android Developer pages [74]. However, the application does not ask for any of the other permissions associated with the usage of Bluetooth and an API level of 31 or higher. These permissions are “bluetooth_scan” used for looking for Bluetooth devices, “bluetooth_advertise” to make the device discoverable to other devices via Bluetooth, and “bluetooth_connect” if the device communicates with Bluetooth devices. The

“fine_location” permission is associated with Bluetooth, and it is needed if the application uses Bluetooth scan results to derive the physical location of the device. Therefore, we observed that it seems like the application was targeting API levels below 31, as you then only need the “bluetooth” permission and “fine_location”. These are needed to perform any type of Bluetooth or BLE communication for Android 11 or lower [74]. The “bluetooth_admin” permission is needed to discover other Bluetooth devices, but it can also be used to manipulate Bluetooth settings. For apps that can run on Android 10 or 11, the “background_location” is also required for discovering Bluetooth devices. This means that if the application targets devices that run on Android 11 and lower, they will ask for both “fine” and “background” locations in order to use the BLE communication on the device.

When it comes to the permission related to battery statistics, this allows the application to see the battery level of the device. In addition, the application is allowed to access information about which applications utilize a large amount of the battery capacity. This again implies that the application can gain access to which application the user spends the most time using [3]. When looking at the capabilities of the Medtronic application on the Play store, it was stated that the application can “ask to ignore battery optimization”. This could mean that the performance of the application will not be affected by the battery preservation mechanisms that the phone uses. However, this capability was related to another permission that the app asked from the user, namely “request_ignore_battery_optimization”. There did not seem to be any specific capability of the application that was directly associated with the “battery_stats” permission that they asked for. This is not a permission that is directly asked of the application user either, as the protection level for this permission is “signature|privileged”. This permission is only granted by the system if the requesting application is situated in a dedicated folder on the Android system image, or if it is signed with the same certificate as the application that declared the permission [2].

6.1.3 Analysis of the APK

After the static analysis of the Medtronic application was performed, we wanted to look further into some of the issues presented by the automated tools. Therefore, we wanted to look into some of the previous findings, to see if these vulnerabilities really are present in the application.

Obfuscation

We wanted to take a look at the code, hoping it would give us information about how the application is structured and give further insight into the potential vulnerabilities found in the static analysis. When downloading the APK file to our computer, this had little value on its own. Therefore, we decompiled the APK file to JAR file, as

described in subsection 4.3.2. When opening the code file inside *jd-gui* we quickly realized that the codebase was obfuscated. This is a common technique used in order to make the code more difficult to understand for a third-party actor. The goal of obfuscating is to make reverse engineering of the application hard to perform.

For Medtronic’s case, the names of the files themselves, as well as the function and method names, were changed to strip them of their meaning. The changed names had received non-alphanumeric characters, which made them hard to understand. In addition, it seemed like the control flow is obfuscated, so it is harder to follow. This can be done by adding loops or conditional statements or changing the order of execution of the functions. Both of these techniques are common when obfuscating code. We discovered that the MyCareLinkHeart app uses Arxan Technologies for its code obfuscation, by looking at the BeVigil analysis. Arxan was an industry leader in the field of application protection solutions and has recently joined forces with other businesses within the security field to create the company Digital.ai [61]. They shield applications from reverse engineering and tampering by using patented technologies. In addition, they provide cryptography solutions to make sure that the keys of the applications remain secure [6].

Digital.ai offers several different products, with some specifically aimed at application security. Applications that are protected by their security measures contain obfuscated machine code that aims to be not understandable by threat actors. In addition, the technology detects when the app is run in an environment that allows it to be tampered with, such as on an emulator, a rooted device, or with a debugger attached. Lastly, it is possible to detect when the code in the application has been modified. Digital.ai also offers the developers visibility into attacks on the application and attempts to run the app in unsafe environments. Their products also provide automatic responses to threats in real-time by forcing a higher level of authentication, changing app features, or shutting down the application under attack.

Restrictions

An attempt to open the application in an emulator was done to see if there were any vulnerabilities despite our problems with the obfuscation of the code. After installing the app on our Android Studio emulator, it was not possible to open the application. It shut down automatically upon trying to open it. We consulted the Medtronic web pages and found that the list of supported devices only consisted of Samsung phones. We had tried with a Nexus phone, and this was assumably the reason for our struggles. It was not possible to emulate a Samsung phone in Android Studio, and thus we had to search for a different emulator.

For the different options we found for Samsung emulators online, there were free trials that lasted from 30 mins to one hour. This was too short of a time limit for us

to be able to execute the analysis we wanted on the Medtronic application. There was also a possibility to pay for access to the emulators, but one of the goals of our thesis was to see what was possible to achieve with free tools. Therefore, we chose not to pay for access to one of these emulators. As we were able to root our emulator, there was a possibility to trick the phone to think it was a Samsung, by exploiting the settings of the phone. This was however not helpful for opening the application either.

After looking further into the information about the application security measures of Digital.ai, an additional reason for us not being able to open the application could be that we tried to run it on an emulator. Combining this with the phone not being a Samsung could be the reason it stopped automatically whenever we opened it. In the BeVigil analysis, it was stated that the application checks both `Build.MODEL` and `Build.PRODUCT` to see if the application was opened on a suitable device. There was also a possible VM check present. In order to gain our understanding of how these checks were used, we had to try to find them in the code.

We tried to find where the check for whether the phone was a Samsung or not was put into play by looking into the code. In addition, we wanted to check if the app had implemented any checks for the use of an emulator or a debugger. The reason for only supporting some device types could be that the manufacturer relies on some specifications that are not implemented in all devices, that are needed to certify the device or application. On the other hand, if there are checks for emulators or debuggers then it is considered a security mechanism. This can be put into place to make sure that it is hard to gain access for alternative purposes than a “normal” patient, for example, to analyze the app. Therefore, the check for whether a device is a Samsung can be related to the functionality of the device, while checks for emulators/debuggers are more present for security.

In the file *ManufacturerUtils*, located within `com/google/android/material/internal`, there is a method called *isSamsungDevice*. This method checks if the phone is a Samsung phone, by reviewing the *Build.Manufacturer* and returns a boolean value depending on whether this value is equal to a predefined string. We were not able to find anywhere in the code where this method is called upon. The *isSamsungDevice* method is using a support function, with a obfuscated function name. Due to the obfuscation of the application and the unusual characters used for the function names, it was not possible for us to search through the code base to see if this support function was called upon anywhere else. Therefore, we were not sure if this function is called otherwise in the code, although we assumed that this is the case.

```

Process crashed: java.lang.NoSuchMethodException: java.lang.Runtime.exec []

***
FATAL EXCEPTION: Thread-7
Process: com.medtronic.crhf.mclh, PID: 11544
java.lang.NoSuchMethodException: java.lang.Runtime.exec []
    at java.lang.Class.getMethod(Class.java:2072)
    at java.lang.Class.getDeclaredMethod(Class.java:2050)
    at kd.йр. (Unknown Source:963)
    at java.lang.reflect.Method.invoke(Native Method)
    at kd.Дл.义 (Unknown Source:305)
    at kd.ўп. (Unknown Source:579)
    at java.lang.reflect.Method.invoke(Native Method)
    at kd.у.р (Unknown Source:52)
    at kd.у.р.run(Unknown Source:0)
***

```

Figure 6.2: Fatal Exception error when using Frida

Frida

We also wanted to try to open the application while using the tool Frida, as described in section 4.3.3. By utilizing Frida, we could achieve a better understanding of how the application looks for the different checks before opening the app on the device. If we understood the flow of the opening process of the application, we could try to dynamically bypass these checks. With Frida, you can pass a Python script to the running application, that can change the normal control flow by overriding the response whenever you arrive at a particular function or method. Seeing as we know that the “isSamsungDevice” function exists, we wanted to try to see if this was called, and thereafter change the response to “True” to override the check. To perform this, we wanted to first run the application with the following command:

```
1 frida-trace -U -i "isSamsungDevice" com.medtronic.crhf.mclh
```

Here, the `-U` means that we connect to a remote device, the `-i` specifies the function to look for, and the last argument is the name of the APK for the application. After executing this we wanted to write a script that changed the response of the function, to pass through this check on the device. However, as mentioned, the Arxan obfuscation of the application also includes checks to see if the application is run in an unsafe environment. This also includes a code instrumentation toolkit such as Frida. Whenever we tried to run the application with the Frida functionality attached, we received a *Fatal Exception* message, with a `NoSuchMethodException`. We assumed the reason for this was that there were checks in place to prevent the application to be run with Frida attached. We tried to follow the function order included in the error description, without being able to figure out what caused the exception. A screenshot of the exception can be found in Figure 6.2. We were thus not able to run the application with Frida and attempt to bypass the checks for Samsung and the usage of an emulator, despite multiple endeavors.

Certificate Details

As discussed in section 6.1.1, the certificate uses the SHA-1 and SHA-256 algorithms. SHA stands for Secure Hashing Algorithm and has two primary use cases in Android applications; data integrity and digital signatures. For data integrity, the hash value behaves as a fingerprint for the data to ensure integrity. This way it is possible to discover tampering attempts on the sensitive data. The SHA-1 hash function generates a 160-bit hash value from the input value, while SHA-256 generates a 256-bit hash value [58].

SHA-1 is a cryptographically broken hash function and is no longer considered secure, but it is still used to this day. It is vulnerable to so-called collision attacks. A collision occurs when two input values, such as two distinct documents, generate the exact same hash value. In practice, a secure hash function would never produce a collision, but the SHA-1 algorithm has this security pitfall [58]. It has been demonstrated a collision attack with SHA-1, known as “SHattered” [70]. This vulnerability can be exploited to compromise the principles of a cryptographic hash function, namely integrity, and authenticity. As the SHA-256 has a much larger output space, it is more resistant to collision attacks.

To confirm the use of SHA-1 and SHA-256, we used the **keytool** command which prints out certificate details of the APK, check Figure 6.3 for the result. As seen, the certificate shows both SHA-1 and SHA-256 fingerprints. It includes two alerts, where the first warning is identical to the one found in the MobSF analysis. The second warning related to “Subject Public Key Algorithm” informs about the algorithm used to generate the public key associated with the certificate [88], which says 1024-bit RSA key. NIST recommends an RSA key size of a minimum of 2048 bits [10]. The computing power has increased which means that it has become easier for an attacker to break encryption faster and more efficiently. Therefore, it is recommended to use a key with larger size, such as 2048-bit or higher, to ensure a high level of security. Additionally, it is crucial that the industry should migrate to SHA-256 or stronger hash functions for safer hashing as soon as possible.

6.1.4 Structure of the Mobile Application

Even though we were not able to analyze the code directly in Medtronic’s application, we were still able to make some observations on how the application is structured. This gave us some insight into possible vulnerabilities within the app, by looking at common pitfalls within the components that Medtronic uses.

```

████████████████████ % keytool -printcert -jarfile MCL_Heart_base.apk
Signer #1:

Certificate #1:
Owner: OU=Ent Mob, O=Medtronic
Issuer: OU=Ent Mob, O=Medtronic
Serial number: 4e00b04d
Valid from: Tue Jun 21 16:53:01 CEST 2011 until: Sun Oct 22 16:53:01 CEST 3009
Certificate fingerprints:
  SHA1: E2:D9:90:BD:48:94:C5:8C:74:B6:C3:E2:39:AB:38:9D:E1:22:70:5A
  SHA256: ED:64:B4:70:53:31:7E:C0:F4:99:9B:F9:49:AA:D1:0F:1D:63:75:B1:CB:03:DD:5D:D7:BC:E3:E2:7D:2E:00:29
Signature algorithm name: SHA1withRSA (weak)
Subject Public Key Algorithm: 1024-bit RSA key (weak)
Version: 3

Warning:
The certificate uses the SHA1withRSA signature algorithm which is considered a security risk.
The certificate uses a 1024-bit RSA key which is considered a security risk. This key size will be disabled in a future update.

```

Figure 6.3: Certificate details of Medtronic

Dagger

Dagger is an automatic tool that helps with managing dependencies in the application. It automatically generates code that imitates the dependency code that otherwise would have been handwritten by the developer. At build time, Dagger builds and validates dependency graphs throughout the whole application. In order to do this, Dagger makes sure that every object's dependencies can be satisfied, to avoid runtime exceptions, and checks for dependency cycles, to avoid infinite loops [23]. In order to use Dagger correctly, you need to add keywords to the constructor of the different objects, so it knows how to create the instances and what their dependencies are. One potential issue with using Dagger is that you need to pay extra attention to whether your external modules are secure, as Dagger automates the process of dependency handling. This is because if the application uses any untrusted modules, these can inject dependencies that give access to injecting code into the application or accessing unauthorized data.

UrbanAirship

UrbanAirship is a US-based company that provides its clients with marketing and branding. They specialize in Mobile App Experience (MAX), where they help companies with increasing app engagement with users of the application. Their goal is to help brands with keeping their customers throughout the whole life-cycle of MAX [55]. For the Medtronic application, UrbanAirship is stated as one of the Analytics Trackers of the device. When we looked at the files associated with UrbanAirship in the code base, we observed that they seem to offer a broad range of services, such as analytics, remote data handling, logging of events, and information about application metrics. UrbanAirship was also mentioned several times during the static analysis, most importantly the analysis highlighted the use of an implicit broadcast in these files.

Communication Protocol

According to Medtronic's websites, the application has the possibility to request a connection to the pacemaker, if the connection is lost [52]. This suggests that there is a two-way communication channel between the application and the pacemaker. Consequently, this implies that the pacemaker is able to accept incoming messages from the application. If this is the case, it could allow an attacker to adjust settings or perform remote monitoring of the pacemaker. However, Medtronic's web pages do specify that it is not possible for any non-BlueSync device to connect with the pacemaker [52]. This means that if a device wants to connect to the pacemaker, it must support BlueSync. In addition, it remains unclear whether a connected device is able to send commands other than connection requests to the pacemaker. To understand the likelihood and possibilities of such a connection occurring, we had to understand the BlueSync protocol.

On Medtronic's own website, they have stated that the BlueSync technology consists of three components. The basis for the communication protocol is BLE, in order to communicate with phones or tablets that are BLE compatible. In addition, to make communication more secure, BlueSync uses an encryption module where the data is encrypted using NIST standard encryption. There is no additional information about specifically which type of encryption is being used, but they assure end-to-end encryption by encrypting the data before it is sent from the pacemaker. Lastly, the protocol uses a high-density integrated circuit to reduce the current drain for increased longevity of the device [50]. Further, there exists some supplementary information about the security measures put in use to protect the device and the data. The pacemakers do not have any IP addresses and are thus not connected to the internet. One reason for this might be to increase the security of the devices, to make it harder for an attacker to connect to the IMD.

There was no specific information available about the details surrounding BlueSync on the website of Medtronic. For example, there was no information regarding whether the BlueSync is continuously on for the pacemaker, or if it is a periodic type of communication. In addition, there was no concrete information about what type of encryption was used for the communication. Therefore, it could be hard for an arbitrary device to be able to connect to the pacemaker. However, as the pacemaker must be able to accept new connections, it is not impossible for an attacker to connect to the device, if they are able to use BlueSync in the correct way.

6.1.5 Summary of our Findings

After performing the analysis for the MyCareLink Heart application, we had been able to get some findings. The analysis of this application had been characterized by the obfuscation tool used by Medtronic. Seeing as this obfuscation made it impossible

for us to open the application with the tools we had available, it was very difficult for us to further look into the things that were found in the static analysis, performed by MobSF and BeVigil. In addition, the obfuscation made it more difficult to understand the code base of the application and identify vulnerabilities in this way. Although the obfuscation made it difficult, we still made some discoveries that are summarized in the table below.

Table 6.3: The summary of findings for MyCareLink Heart App

| Discovery | Explanation | Covered In |
|----------------------------------|--|------------|
| SHA-1 for certificate signing | The application uses SHA-1 with a 1024-bit RSA key for signing the certificate for the application. This is a weak signature algorithm and a weak key length. | 6.1.3 |
| Dangerous Permissions | The application asks for the battery statistics of the device, seemingly without any proper reason. | 6.1.2 |
| Dagger for managing dependencies | Using Dagger requires the developer to be sure of the security of external modules, as these modules can inject untrusted dependencies into the application without the developers' knowledge. | 6.1.4 |
| Arxan for obfuscation | The app's obfuscation methods include: changing method and function names, encrypting strings in clear text, increasing the complexity of control flow, checking to see if the app is run in an unsafe environment, and probably inserting arbitrary code. | 6.1.3 |

6.2 Patient App - Biotronik

The Patient App for Biotronik is an addition to the existing ecosystem, designed to give patients more insight into their own health. The application does not communicate directly with the pacemaker or the HMU. The application gives the patient information about the status of their device and its battery, as well as information about their heart rhythm. In addition, the patient can log their own symptoms, which can help the responsible doctor keep track of the overall health of the patient [65].

In the analysis of the Patient App by Biotronik we first present the results from the static analysis, performed by MobSF and BeVigil. Afterward, we dive deeper into the permissions requested from the user by the application, before we look further at the details of the certificate for the application. Thirdly, we go through the structure

of the application, which we found by looking through the APK and dynamically running the application. The next section includes bypassing the authentication of the application, both for registering a new user and for the login of an existing user. Lastly, we present some unverified issues, before we summarize the most important findings of our analysis.

6.2.1 Static Analysis

For the static analysis of the Patient App from Biotronik, we used two tools, MobSF and BeVigil. The Patient App received an overall security score of 50 of 100 from MobSF. This meant that the application was within the Medium risk level, described in the section 4.3.3. For the analysis performed by BeVigil however, the application gained a score of 9.1 out of 10, i.e. 91 out of 100 [36]. Again, we observed an enormous difference between the two analyses, and we needed to look further into these. There were quite a few security issues with this application found in both analyses, which we run through in the following sections. The categorization from MobSF is used as the basis for our analysis, and we add information from BeVigil to the categories, as well as additional information at the end. The results from the BeVigil analysis are gathered from [36].

Application Permissions

Several permissions were categorized as *Dangerous* by MobSF in the analysis. The same permissions were marked as unsafe by BeVigil, but here with the level of *Risky*, which is the level below on the scale. Both analyses agreed on which permissions they found troublesome. These permissions are:

- android.permission.ACCESS_FINE_LOCATION allows the application to access fine location sources, giving the most precise location
- android.permission.CAMERA allows the application to take both photos and videos with the phone’s camera.
- android.permission.READ_EXTERNAL_STORAGE allows the application to read from the external storage of the phone
- android.permission.WRITE_EXTERNAL_STORAGE allows the application to write to the external storage

The location permission makes it possible for the application to receive the most accurate location information about the phone, and is a runtime permission, as in the case of Medtronic. Next, the permission to use the camera of the phone makes it possible for an application to collect images through the camera at all times. This

means that the application can access the camera without the user's knowledge. The permission asked for in this application is actually a deprecated version of the Camera permission. This gives the application permission to use the phone's camera for pictures and videos [75].

When it comes to the permissions regarding the external storage, these imply that the application has access to read and modify other parts of the mobile phone's storage. On older versions of Android, both `READ` and `WRITE_EXTERNAL` were required to access any file outside of the app-specific directories [76]. These permissions can be connected to the Camera permission, as it is common to ask for the `WRITE_EXTERNAL_STORAGE` permission in order to be able to save the images taken by the application to the regular folder for pictures. There could also be a connection to the location permission, as the location is required in order to be able to tag the images taken by the application with the GPS location information [75].

Certificate Analysis

There was only one warning present in the certificate analysis of the application done by MobSF. This warning was the same as the one for Medtronic, that the application is vulnerable to Janus Vulnerability. As mentioned above, for the analysis of the Medtronic application, this was a warning as it is only applicable for some versions of the Android OS, and for some signature schemes. More details about this vulnerability are already discussed in section 6.1.1. Similar to Medtronic, some of these threats are analyzed and discussed afterward in subsection 6.2.6.

Manifest Analysis

The analysis of the application's manifest resulted in only warnings for the Biotronik application. In total, there were four warnings in the analysis by MobSF, while there are two warnings in the BeVigil analysis. For the MobSF, two of the issues were related to shared broadcast receivers. As for the Medtronic application, the receivers are protected by permissions, but it is not clear in the analysis of the application which level of permission is enforced. This specific issue was discussed in more detail previously, in section 6.1.1. A similar type of issue was also highlighted in another warning, involving a service shared with other applications on the device. As similar, there was a permission for protection with an unknown level, potentially allowing access to the service by other apps. Once again the permission for the service was not known for the analysis, indicating that it may have already been adequate. The permissions for this service should be checked.

In addition, MobSF focused on the possibility to back up application data. This is seen in the manifest.xml file through `android:allowBackup=true`. This was a warning because this allows an attacker to back up the application data via the

ADB. This means that users who have enabled USB debugging are able to copy the application data of the device. This permission can be positive, as it allows backup and restoration of the application data by the ADB, for example in the case of a full-system backup. On the other hand, it allows for the backup of application data to malicious devices through a USB.

The BeVigil analysis had a bit of a different focus area, and the two reported issues with severity medium for this analysis were both related to exported activities. This means that there are two different activities in the manifest file that has *android:exported=true*. This can be dangerous as exported activities can be run by third parties, by directly prompting this activity to open when running the application. This can possibly result in bypassing authentication mechanisms that are in place in the application. This can happen by calling the activity with the exported option during the opening process of the application, so it never goes through the authentication page.

Code Analysis

The code analysis in Medtronic in MobSF is equivalent to the Biotronik one. Therefore, we refer to Table 6.1 for Biotronik. The explanation of the different issues is given in section 6.1.1. There were some extra vulnerabilities mentioned in the BeVigil analysis for Biotronik. Most of them are similar to the ones mentioned for the Medtronic analysis, which have been discussed in Table 6.2. There was one additional vulnerability in the analysis of the Biotronik application, which can be seen in Table 6.4.

Table 6.4: The result of the code analysis by BeVigil of Patient App

| ID | Severity | Info | Standards |
|----|----------|---|---|
| 10 | Medium | The app uses weak crypto algorithms, which can allow an attacker to break the ciphered communication and gain access to plain text content. | CWE: CWE-327: Use of a Broken or Risky Cryptographic Algorithm |

The new issue is related to the usage of a weak cryptography algorithm. This issue is found in four different files in the application, all within the Google folder. The crypto algorithm in use is MD5, which is not recommended to be used for cryptographic authentication, according to IETF [85]. MD5 is a cryptographic hash algorithm that takes an input of any length and changes it into a fixed-length message of 16 bytes. MD is an abbreviation for “message digest”. It has been considered insecure for a number of years, due to the hash collisions it causes as well as how easy it is to reverse the encryption [91]. This implies that the app uses the MD5 algorithm

and that the algorithm is utilized within the app. Using the MD5 algorithm for cryptographic purposes makes the app vulnerable to various attacks, such as collision attacks and rainbow table attacks. This is further discussed under Unverified Issues in section 6.2.7.

Shared Library Binary Analysis

For the shared library binary analysis of the Biotronik application, BeVigil did not find any issues that were noted. MobSF found two issues with a severity high and two issues with a severity of warning. The two warnings with severity high concerned canary stack values and the setting of the NX bit. For the canary stack, this was relevant to about half of the shared libraries used in the application. As mentioned in the analysis of the Medtronic app, stack canaries are used to detect and prevent exploits from overwriting return addresses. The NX bit stands for a Non eXecutable bit, and the issue was that the library did not set the NX bit. This bit offers protection against memory corruption vulnerabilities. These exploitations can be avoided by marking a memory page as non-executable, which is done by setting the NX bit to true [59]. This issue was also relevant to around half of the libraries, and no shared library had both of these issues simultaneously.

For the two warnings with severity medium in the shared library analysis, the issue was regarding the fortification of functions and the stripping of symbols. Both of these are relevant for all the shared libraries of the application. The fortification issue is regarding the lack of fortified functions, which are functions that provide buffer overflow checks against common insecure functions. The functions are part of glibc, which is an acronym for GNU C libraries, the core libraries for the GNU system [82]. The issue of stripping symbols is related to the fact that there are symbols available, but these are not stripped. This means that the symbol table and debugging information contained in the native libraries are not removed. Stripping of code libraries results in significant size savings. On the other hand, stripping makes it impossible to diagnose crashes on the Google Play Console, as too much information is missing [77]. As mentioned these issues are present in all the shared libraries in the analysis, which can indicate that this is something that should be looked further into.

Additional Information from BeVigil

We introduce some of the findings from the categories that have not yet been discussed. The BeVigil analysis reported a string issue with the severity medium. The vulnerability was regarding a possible secret being detected, with sensitive credential information. The issue can also be found in Table 6.1 with ID 6. This issue was found in four different files, where three of them were situated in the *strings.xml* file in the resources folder, and the last one was present in the *Configuration.json* file

in the same folder. As similar to the analysis in Medtronic in section 6.1.1, two of the three strings within *strings.xml* are associated with Google API keys, while the last concerns a password content description. One particular string issue with the description “Sensitive credentials information detected” caught our attention. The string is a ClientSecret found in the *Configuration.json* file. The implications of this string are discussed further in subsection 6.2.5.

BeVigil also analyzed the assets associated with the application, without finding any vulnerabilities here. There was no detected malware in the application either. When it comes to the trackers present in the application, there were four results. Two trackers were used for analytics, Google Firebase and Microsoft Analytics, and one was used for crash reporting, Microsoft Crashes, and lastly, we had one tracker used for advertisement, which is Google AdMob. This gave us some indication on which companies receive the data from the application, and how much data was collected.

In addition, the BeVigil analysis gave some information about the *classes.dex* file, which in Biotronik’s case was split into *classes.dex* and *classes2.dex*. There was no information present about obfuscators used on the code, which we also noticed when opening the code to study it. The analysis also revealed that the application had included several checks that go into the anti-VM category, such as Build.FINGERPRINT, Build.MODEL, Build.MANUFACTURER, Build.PRODUCT, Build.HARDWARE, as well as checks on the SIM operator and network operator name. This indicates that the application pays attention to what kind of device it is run on, and that performance of the application can be influenced accordingly. We look into where these checks are being used, to determine the purpose.

6.2.2 Permissions

We discovered information about some dangerous permissions in the static analysis performed by MobSF and BeVigil. We were interested in further investigation into these vulnerabilities to see how the permissions were used in the application.

External Storage Permissions

The application asked for permission to both read and write to the external storage of the device. As mentioned in the section 6.2.1, the permission associated with writing to external storage could be because of the Camera permission, where the user got the opportunity to save the photos taken to the device’s camera roll. After looking through the files in the *classes.dex* file of the Biotronik application, we found several ways that the application accesses external storage.

Firstly, the application uses a method called `getExternalStorageDirectory()`, which is not recommended anymore. This was used to access external storage on older

versions but is now deprecated. The returned directory is a storage that is shared across all applications on the device [27]. In addition, the method `getExternalFilesDir()` is used to store private data specifically for the app only. Before this method is used, there is a check to see if the SDK is larger than 19. For these external storage files, there is no security enforced. This means that any application with the permission “write_external_storage” can write to the files [18]. The last method that is frequently used for external storage handling is `getExternalCacheDir()`. This method gives the absolute path to an application-specific directory on the primary storage, where the app can store cache files. These files are internal to the application and typically not visible to the user. It is important to note that these files are not always monitored for available space, which means that files might be automatically deleted [18].

Camera Permission

As determined in the static analysis, the application also asks for permission to access the camera of the device. When looking through the files of the `classes.dex`, we observed that there were two versions of the camera library being used in the application. As mentioned in section 6.2.1, one of these versions is deprecated today. This is the “Camera” library. There is however also another library in use, which is the “Camera2” library. This library provides in-depth control for complex use cases and requires the developer to manage device-specific configurations [75].

We observed that in `classes.dex` and `classes2.dex`, there were several files that include the Camera or Camera2 libraries. The files situated in `classes2.dex` are the ones that utilize the deprecated version of the Camera library. These files are listeners, that get notified of face detection, by the method `Camera.Face[]`, and zoom changes, by utilizing `OnZoomChangeListener`. In addition, there is a `CameraEventListener` in `classes2.dex`, that looks for changes in focus and frames, as well as capturing frames from the camera sensor. In `classes.dex` there is only one file that uses the camera2 library, where there is a `CameraManager` present.

The `onZoomChangeListener` is a public method that is called whenever the zoom value changes during a smooth zoom. The `FaceDetectionListener` notifies the listener of faces detected in the frame from the camera sensor. The `CameraEvent` class uses a `PreviewCallback` function, which captures frames from the camera in real-time. In addition, the class uses an `AutoFocusCallback` function, which is called when the camera completes the autofocus. This event listener is notified whenever the camera captures a frame or the focus is set on a person or object. Therefore, this file contains the main functionality for the camera usage of the application. All the functions and interfaces presented in this paragraph are deprecated, and not recommended to use by the Android Developers pages [66].

Location

For the location permission, the application only asks for “fine_location”, which is the most accurate location of the mobile device. This location is given by the GPS signal of the device, and can thus determine the location quite accurately. When asking for this location level, the application also implicitly declares a need for foreground location. This means that the location is requested because of an activity within the application when the app is situated in the foreground. If the application wishes to access the location while the app is running in the background, there is a separate permission that is needed - “access_background_location”. However, the application is only required to ask for this for mobile devices running Android 10 (API 28) and higher, for older versions background location is automatically received when the foreground location is accessed [47].

According to the Android Developer pages, a common reason for an application to ask for background location is the usage of Geofencing API. Geofencing combines knowledge of the user’s location with knowledge of locations that might be of interest. You can define geofences in a specified radius around a point of interest, and define different characteristics within each “fence” [19]. When looking through the files of the classes.dex, we saw that the Geofencing.API was present in the Biotronik application. In addition, there are files such as GeofenceBuilder, that create new Geofences, and GeofencingRequest that sets how geofence events are triggered. Lastly, there is a Geofence Transition Receiver present among the broadcast receivers specified in the manifest file. This means that the application contains all the necessary files to create geofences, monitor the users’ location in accordance with the geofences, and trigger related events whenever a user enters a geofence [19].

6.2.3 Structure of the Mobile Application

After the execution of the static analysis, we wanted to dive further into the issues found by the automated tools. For the manual analysis of the application, we inspected the code in jd-gui, as described in subsection 4.3.2. We navigated through the code and expanded the different packages, classes, and methods to get an overview. We searched for different keywords, field names, constructors, strings, and method names to analyze the code more specifically. We highlighted suspicious and interesting codes for further investigation. Furthermore, we set up proxies as explained in section 4.3.3. This allowed us to uncover various insights regarding the structure and components of the mobile application.

Communication Protocol

By analyzing the packets in Wireshark, it was determined that the Biotronik app used TLS. Refer to Figure 6.4 to see the captured HTTPS-traffic with TLS. TLS is a

| No. | Time | Source | Destination | Protocol | Length | Info |
|-----|-----------|-----------|-------------|----------|--------|--|
| 61 | 33.566979 | 127.0.0.1 | 127.0.0.1 | TLSv1.. | 203 | Client Hello |
| 63 | 33.622003 | 127.0.0.1 | 127.0.0.1 | TLSv1.. | 150 | Server Hello |
| 65 | 33.624064 | 127.0.0.1 | 127.0.0.1 | TLSv1.. | 1944 | Certificate |
| 67 | 33.634377 | 127.0.0.1 | 127.0.0.1 | TLSv1.. | 361 | Server Key Exchange |
| 69 | 33.638453 | 127.0.0.1 | 127.0.0.1 | TLSv1.. | 65 | Server Hello Done |
| 71 | 33.650935 | 127.0.0.1 | 127.0.0.1 | TLSv1.. | 149 | Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message |
| 73 | 33.662434 | 127.0.0.1 | 127.0.0.1 | TLSv1.. | 141 | Application Data |
| 75 | 33.663264 | 127.0.0.1 | 127.0.0.1 | TLSv1.. | 2192 | New Session Ticket |
| 77 | 33.663803 | 127.0.0.1 | 127.0.0.1 | TLSv1.. | 62 | Change Cipher Spec |
| 79 | 33.664856 | 127.0.0.1 | 127.0.0.1 | TLSv1.. | 101 | Encrypted Handshake Message |
| 97 | 40.306676 | 127.0.0.1 | 127.0.0.1 | TLSv1.. | 573 | Client Hello |
| 99 | 40.327681 | 127.0.0.1 | 127.0.0.1 | TLSv1.. | 183 | Server Hello |
| 101 | 40.328752 | 127.0.0.1 | 127.0.0.1 | TLSv1.. | 62 | Change Cipher Spec |
| 103 | 40.329635 | 127.0.0.1 | 127.0.0.1 | TLSv1.. | 135 | Application Data |
| 105 | 40.332896 | 127.0.0.1 | 127.0.0.1 | TLSv1.. | 1984 | Application Data |
| 107 | 40.337947 | 127.0.0.1 | 127.0.0.1 | TLSv1.. | 358 | Application Data |
| 109 | 40.338130 | 127.0.0.1 | 127.0.0.1 | TLSv1.. | 146 | Application Data |
| 127 | 47.797138 | 127.0.0.1 | 127.0.0.1 | TLSv1.. | 573 | Client Hello |
| 129 | 47.805540 | 127.0.0.1 | 127.0.0.1 | TLSv1.. | 183 | Server Hello |
| 131 | 47.807080 | 127.0.0.1 | 127.0.0.1 | TLSv1.. | 62 | Change Cipher Spec |

Figure 6.4: TLS-traffic in Wireshark for PatientApp, the Biotronik mobile application

robust cryptographic protocol that ensures the secure transmission of data between applications over the Internet [92]. Further, the HTTPS is the secure version of HTTP, and it uses TLS to set up a secure and encrypted connection. To intercept and manipulate Hypertext Transfer Protocol Secure (HTTPS) traffic, techniques such as MITM proxy was employed. However, it is important to note that the security measures implemented in HTTPS/TLS-encrypted communication can make it difficult for the general public to intercept or manipulate the communication between a mobile application and a remote server.

In order for the mobile app to trust the intercepted communication, certain configurations are required. Mobile apps typically rely on trusted certificates and certificate authorities to ensure the authenticity of the server they communicate with. This is where a proxy comes in handy. When using a proxy, we generate a self-signed certificate that mimics the original server certificate. In our case, to facilitate the analysis, we set up a proxy that intercepted and analyzed HTTPS/TLS-traffic, generated a certificate, and made the mobile application trust the certificate by adding it to trusted certificates on the emulator. This allowed us to examine the packets and gain insights into the behavior and potential vulnerabilities of the mobile application.

Authentication

By analyzing the mobile applications network traffic in Burp Suite, we determined which authentication protocol the app was using. The request to validate credentials in Figure 6.5 includes the path “POST /auth/.../openid-connect/...” which means that the application uses OpenID Connect (OIDC) for authentication. OIDC is an authentication protocol built on top of the OAuth 2.0 framework.

Another important tool used was logcat in Android Studio and it provided a log



Figure 6.5: Authentication request in Burp Suite

```

//de.biotronik.PatientApp I/: [Information] Retrieving access token for given refresh token.
//de.biotronik.PatientApp I/: [Information] Initialized FixedCaboAnalyticsService
//de.biotronik.PatientApp I/: [Information] Failed to get access and refresh token for given user: StatusCode=Unauthorized, ResponseBody={"error":"invalid_grant","error_description"}
//de.biotronik.PatientApp I/: [Information] logging in as ...com
//de.biotronik.PatientApp E/: [Error] logging in as ...com failed with WrongCredentials
//de.biotronik.PatientApp I/: [Information] Failed to get access token for given refresh token: StatusCode=BadRequest, ResponseBody={"error":"invalid_grant","error_description":"Tn
//de.biotronik.PatientApp I/: [Information] replication auth error trying to get access token based on stored refresh token, but can't retrieve it (error: RefreshTokenInvalid) for D
//de.biotronik.PatientApp I/: [Information] replication auth error trying to get access token based on stored refresh token, but can't retrieve it (error: RefreshTokenInvalid) for D
//de.biotronik.PatientApp I/: [Information] failed to buildReplicationAuthenticator for DeprecatedPush, can't start replication
//de.biotronik.PatientApp D/EGL_emulation: app_time_stats: avg=149.23ms min=3.78ms max=1440.89ms count=10

```

Figure 6.6: Logcat in Android Studio

of system messages. When entering with the wrong credentials in the BIOTRONIK mobile app, it logged “Failed to get access token for given refresh token (...)”, as shown in Figure 6.6. The information provided indicates the use of OIDC since it uses to access and refresh tokens as part of its token-based authentication and authorization mechanism [87]. Additionally, by intercepting the traffic with mitmproxy (section A.7) and analyzing the response, we observed that an authorization header started with Bearer, with an access token we added manually. This again implied that the mobile application is using OIDC for authentication.

OIDC uses JSON Web Tokens (JWTs) as the primary token format to prove that users are authenticated between parties. In OIDC, if a user tries to authenticate (in our case) to a mobile app, the app issues an ID token that is a JWT. A JWT contains information about the authenticated user, such as username. The use of JWT has several benefits, including improved security since they are digitally signed, and efficiency as they are quick to verify [60]. If the JWT is properly implemented and secured, it could be difficult to forge a token. A forged token is a token created by an attacker and appears to be from a legitimate source. The goal is to access protected resources. One of the pitfalls with JWT is insufficient validation checks, such as checking the signature or verifying other fields in the file. Therefore, we tried to utilize a common security vulnerability related to JWT by forging a token and sending it back with mitmproxy.

To bypass access controls it is necessary to understand how a JWT works. A JWT consists of three parts: a header, a payload, and a signature. The header identifies the algorithm that generates the signature, the payload contains the information used to bypass authentication, and the signature is used to validate the token in case of a tampering attempt. There exist different ways to forge a token, and one of them is called “none algorithm”. This means that the **alg** field in the header is set to none, and the signature section of the token is then empty. In this case, any token with an empty signature would be considered valid. The following token that supports a “none” algorithm is taken from the website [42]:

```
eyJAiYWxnIiA6ICJOb25lIiwgInR5cCIgOiAiSlldUiB9Cg.eyJB1c2VyX25hbWUgOiBhZG1pbiB9Cg
```

After setting up a mitmproxy, described in appendix A.5, a Python script was made to intercept requests and make responses based on the hosts and the paths associated with the mobile application. The script is provided in subsection A.7.1 sent with JSON files containing necessary fields for each individual response, such as `token_type`, `access_token`, etc. We assumed that for the `userinfo` path, it was essential to include a `user_id` field in the `userid.json` file since it uses OpenID. As said earlier, we interpreted that the response sent to the authentication server of Biotronik needed an `access_token` and a `refresh_token` from Figure 6.6. This was included in the `accesstoken.json`. Moreover, our fake server added 200 OK success status codes to each response. We checked each system message with `logcat` to observe which response should be modified - this process was iterative.

By creating messages to roughly correct responses, we achieved new information and errors. The only error left was a 401 Unauthorized to the server with path “`patientapp-data.biotronik-homemonitoring.com`”, see Figure 6.7. By looking up the URL, the website displayed a JSON response returned by the Couchbase Sync Gateway. We discuss Couchbase below in section 6.2.3.

Backend Server

Previously, we discovered which type of backend server the application used. The JSON response was the following:

```
“couchdb”:“Welcome”,“vendor”:“name”:“Couchbase Sync Gateway”,“version”:“3.0”,  
“version”:“Couchbase Sync Gateway/3.0.4(13;godeps/) EE”
```

The “`couchdb`” indicates the Sync Gateway is based on the CouchDB database. Next, “`vendor`” provides information about the vendor of the Sync Gateway, namely Couchbase Sync Gateway with version 3.0. Finally, “`version`” tells the full version number 3.0.4, build number 13, and other information. This response is sent when

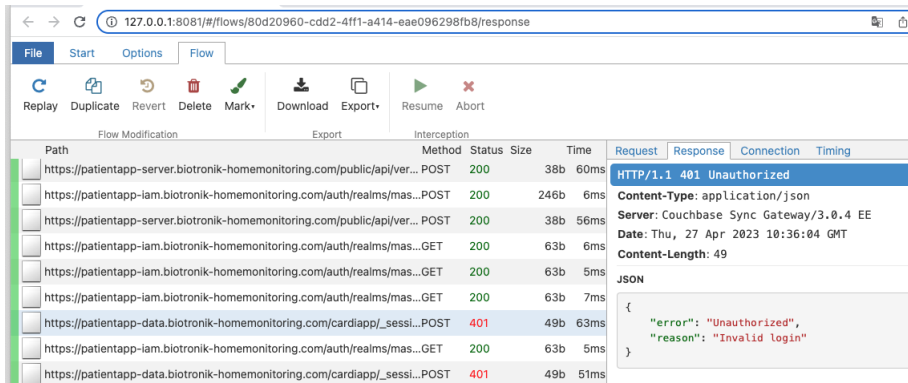


Figure 6.7: The mitmweb interface

making a request to its root endpoint of the Couchbase Sync Gateway, and the server responded with that JSON response. PatientApp uses Couchbase Server to store and manage data that is required by the application and Couchbase Sync Gateway for data synchronization. Couchbase Server is a NoSQL cloud database often used as a backend for mobile applications [80].

During the process of bypassing authentication, we received an error saying “replication auth error unable to get sync gateway session token”, shown in Figure 6.8. Here, we tried to fake responses to these requests as well, but without succeeding. In other words, we attempted to authenticate with the Couchbase Sync Gateway server, but this authentication failed.

HTTP is the primary protocol for communication between clients and the server used by Couchbase Sync Gateway. Additionally, Sync Gateway uses a protocol called Replication Protocol and it provides bidirectional replication of data. This means that the Sync Gateway instances can communicate with each other or with Couchbase servers. Couchbase Server is used to store and manage data required by the application, while Couchbase Sync Gateway handles data synchronization. The protocol is built on top of HTTP. With the replication error, we assumed that we were unable to authenticate to this Replication Protocol, which prevented the login attempt from succeeding. When an authentication attempt fails, the user trying to log in is not able to obtain a session token or establish a replication connection. This prevents data from being replicated.

The newest version of the Couchbase Sync Gateway is 3.1.0. The version used in Couchbase Sync Gateway for the application is 3.0.4, and there are two new versions since this was deployed. We found out that there existed related security

```

<no-tag> de.biotronik.PatientApp I [Information] Validating access token.
<no-tag> de.biotronik.PatientApp I [Information] Initialized FixedCaboAnalyticsService
<no-tag> de.biotronik.PatientApp I [Information] Initialized FixedCaboAnalyticsService
<no-tag> de.biotronik.PatientApp I [Information] replication auth error unable to get sync gateway session token for DeprecatedPull
<no-tag> de.biotronik.PatientApp I [Information] replication auth error unable to get sync gateway session token for DeprecatedPush
<no-tag> de.biotronik.PatientApp I [Information] failed to BuildReplicationAuthenticator for DeprecatedPush, can't start replication

```

Figure 6.8: Logcat in Android Studio for PatientApp

vulnerabilities to that specific version. On Couchbase’s own documentation page², a list of fixed issues for each version is provided. By checking the available release notes for the public for versions 3.0.5 and 3.1.0, one bug fix in version 3.1.0 addressed an issue that affected version 3.0, which includes 3.0.4. The bug has the title “AccessLock not being released when a PUSH replication is ongoing”³ and has the priority as major. This can potentially affect the users of the database who need to take it offline for different reasons, such as maintenance. This issue is fixed in 3.1.0.

Version 3.0.4 have one known issue regarding SSL memcached port. It suggests that Sync Gateway requires all Couchbase server nodes to use the same SSL memcached port. The memcached port is used for communication between the Sync Gateway and the Couchbase port, and if the SSL is not properly configured, it can result in communication failures or unexpected behavior. This is a known issue for the later versions as well. Lastly, the newer versions of the software include bug fixes, performance improvements, and security enhancements, so upgrading to the latest version can be beneficial for the system even though issues are not experienced yet.

Couchbase Server, like any other server, can be configured insecurely. If the best practices for securing are not followed, such as the insecure implementation of authentication or weak password usage, an attacker can gain unauthorized access and insight into data. This security pitfall belongs on the server side, hence this is out of scope and we will not explore this further.

Keycloak

Keycloak is used as an Identity Access Management (IAM) tool to secure PatientApp by providing authentication and authorization services, among other things. Keycloak is integrated with the application by using OIDC, explained in the previous paragraph. We discovered that the mobile application used Keycloak by visiting the web page or endpoint related to the authentication functionality of the app, see the user interface in Figure 6.9.

A significant security pitfall related to Keycloak can be exposing its administration console or API endpoints. The consequence can be providing unauthorized users access to sensitive information and functionalities which compromise the security of

²<https://docs.couchbase.com/sync-gateway/3.0/release-notes.html>

³<https://issues.couchbase.com/browse/CBG-2731>

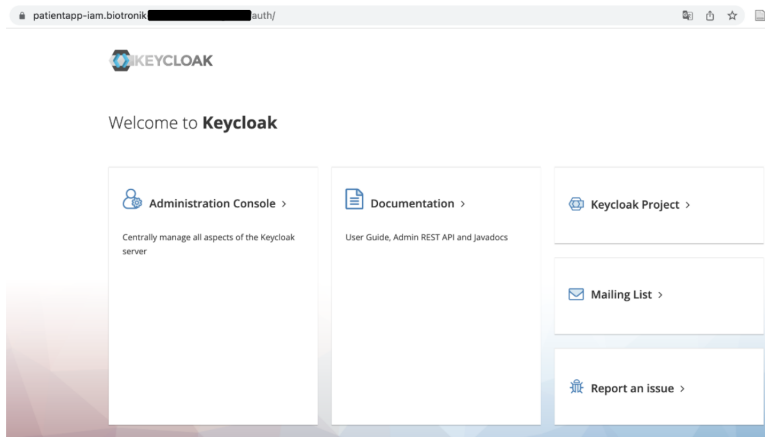


Figure 6.9: One of the server’s websites displaying KeyCloak

the application. We attempted to visit the administration console at `/auth/admin`, but were not able to access it. This does not definitively confirm that the site is secured, but it suggests that access controls may be in place and that restrictions with appropriate security measures are set. However, we were able to access the Keycloak interface from the URL, which made it easy for us to learn that the application uses Keycloak. To increase security even further, users should not be able to access this interface either.

Firebase

The application has integrated Firebase services. This was discovered in the map structure when analyzing the code, in the folder `com > google > firebase`. Firebase is a mobile and web application development platform owned by Google. It provides tools and services to build high-quality applications quickly and easily. It includes services such as authentication, real-time database, analytics, and more [29]. Firebase was used for analytics. In addition, it encrypts data in transit using HTTPS.

Servers

During the application analysis, we obtained four domain names. A web-based DNS client, called `nslookup.io`⁴, was later used to query DNS records for the domain names. By entering the discovered servers, we obtained IPv4 addresses for each of the servers. To acquire ownership information and other relevant details about the IP addresses, we used a tool called IP WHOIS Lookup.⁵ It turned out that Biotronik

⁴<https://www.nslookup.io/>

⁵<https://www.whatismyip.com/ip-whois-lookup/>

```

person:      Guenter Behrens
remarks:    Netadministrator
address:    Biotronik SE & Co KG
address:    Woermannkehre 1
address:    12359 Berlin
address:    Germany
phone:      +49 (0) 30 68905-2410
nic-hdl:    GB6809-RIPE
mnt-by:     VT-MNT
created:    2008-05-28T07:30:55Z
last-modified: 2010-03-24T11:39:39Z
source:     RIPE # Filtered

person:      Thoralf Freitag
remarks:    Netadministrator
address:    Biotronik SE & Co KG
address:    Woermannkehre 1
address:    12359 Berlin
address:    Germany
phone:      +49 (0) 30 68905-4611
nic-hdl:    TF1653-RIPE
mnt-by:     VT-MNT
created:    2008-05-28T07:27:45Z
last-modified: 2010-03-24T11:39:38Z
source:     RIPE # Filtered

```

Figure 6.10: IP WHOIS Lookup of one of the domain names of Biotronik

owns the IPs of the servers. Other details were also provided, such as the address, the owner’s name, and the owner’s phone number. Figure 6.10 is a brief excerpt taken from the website displaying ownership information about the IP address related to the couchbase server.

A sequence diagram of the authentication process between the user and the servers is provided to achieve an overview of how the servers are used in the Biotronik system, see Figure 6.11. This sequence diagram shows a user trying to log into the mobile application, how the servers are communicating with the mobile application, and what type of fields and messages are contained in each request and response.

As we can observe in the sequence diagram, the device first sends a message to validate the version of their application API, to the patientapp-server. This server thereafter responds with a “Version supported”. The next step for the device is to send the user login information to the patientapp-iam server, containing the e-mail address and password of the user. The IAM server then responds with an access and refresh token to the device, which it uses when communicating with the servers, to show that it is authenticated. The device also sends some information about the device, containing versions of OS, information about the application, and other valuable insights to the analytics server, but this is not a part of the authentication. After receiving the refresh and access token, the device tries to validate this refresh token by sending it to the IAM server again. We responded with a 200 OK and a re-sending of a new access token. According to the OAuth

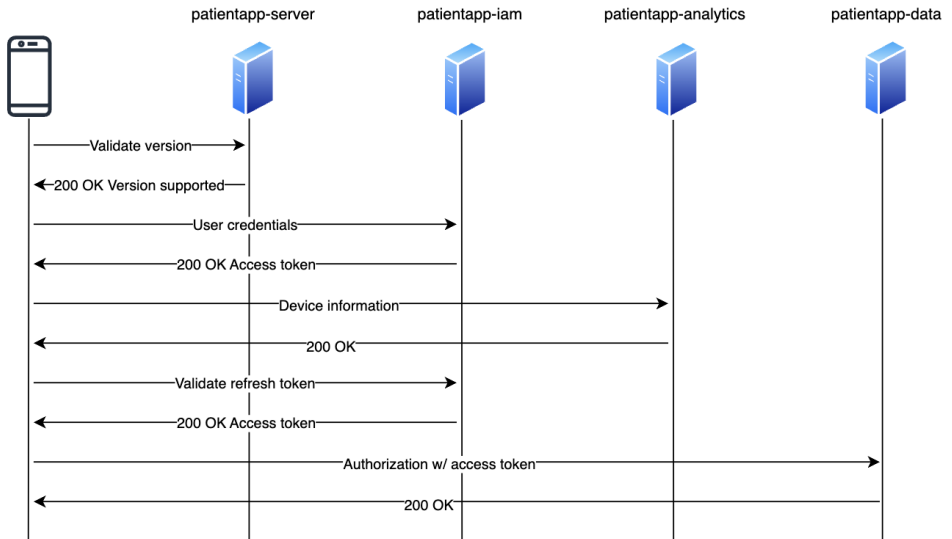


Figure 6.11: A sequence diagram showing a device trying to log in to the application, with communication with the servers in Biotronik

2.0 specifications, the authorization server was supposed to issue a new access token and an optional refresh token back to the client [34]. We did not include a refresh token. Lastly, the device tries to authenticate with the received access token to the patientapp-data server, which is the server actually containing the patient data from the user. However, it appears that the login attempt failed due to incorrect authentication to the Couchbase.

6.2.4 Client Bypass Authentication

We attempted to register a user and bypass regular login, after setting up a mitmproxy, as described in section 4.3.3. By attempting to bypass authentication, we assessed the effectiveness of the authentication mechanism and identified some potential vulnerabilities that could be exploited. The requirement for registration was to possess a legitimate pacemaker as you need to provide a serial number and production date of the pacemaker during the procedure. The goal was to register without sending the request to the actual server, but to trick the mobile application into thinking that the user was registered properly. Further, we wanted to bypass the login and gain unauthorized access to determine if there exist any security vulnerabilities that

could potentially be exploited by attackers. In addition, we wanted to validate the application's security controls.

Register User

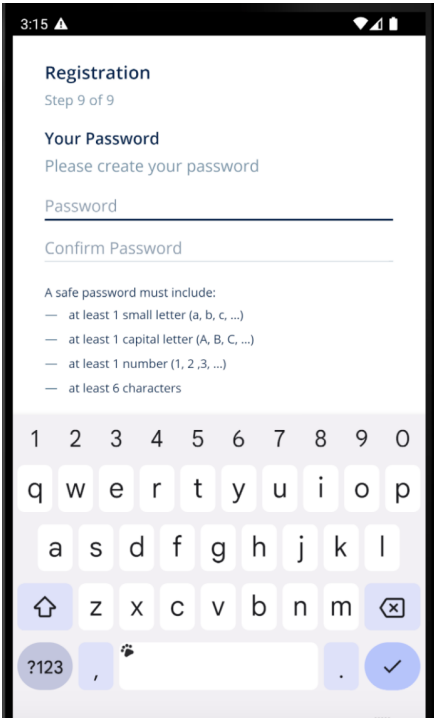
We successfully tricked the application into thinking we registered a user, the receipt is shown in Figure 6.12. This was done without a pacemaker ID, by responding back to the mobile application with multiple 200 OK messages. This was done by constructing a response, visit section A.7 for the Python script and the JSON file named `register.json`. When going through the registration process on the device, the app tried to validate if the user inputs valid information at several points. First, the patient's pacemaker serial number and HMU serial number were checked for validity. This request went to "patientapp-server", which was intercepted and we faked a 200 OK response. After this check, the user was sent to the final step of the registration, which was a generation of a password for login. The password criteria are shown in Figure 6.12a and a confirmation of a successfully registered user for the Biotronik Patient App. This success message was obtained by intercepting the request going to the server and answering with a 200 OK. The application thus thought that the user was registered correctly with the server and that all information that had been inserted was valid. We wanted to trick the application into registering a user to the app, in order to observe the different checks on the client side during the process. In addition, this strategy was followed to find the password criteria for the application.

Login User

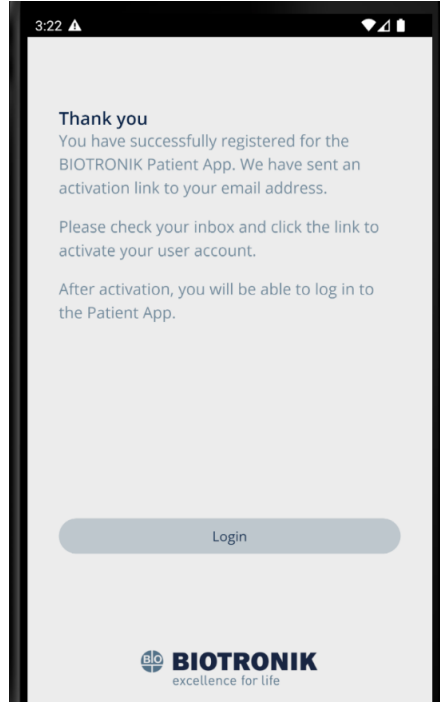
We have discussed the steps to "fake login" as a user already, during the explanation of how the application is structured in Figure 6.11. After several attempts of constructing responses, we successfully sent 200 OK messages to all the requests from our fake server, as seen in Figure 6.13. This was accomplished by intercepting all the requests and creating responses with necessary fields in the body, shown in section A.7. However, the content of the body is still somewhat uncertain. We achieved an error replication error, shown in Figure 6.8. As explained in section 6.2.3, we tried to authenticate with the Couchbase Sync Gateway server. This gave replication errors since the credentials provided were invalid, or the Couchbase database was protected. We can observe in Figure 6.13 that multiple exact get-requests are sent to path `/userinfo`, as the requests contain the wrong information. To conclude, our forged tokens with fake credentials would not bypass authentication.

6.2.5 Application Authentication

As mentioned in section 6.2.1, we found a string in the `Configuration.json` file that was a possible secret. The string is called `ClientSecret` and is shown in Figure 6.14. This vulnerability is related to ID 6 in Table 6.1. When looking through the file



(a) The password criteria



(b) The message in the mobile application of a successful registration

Figure 6.12: The final process of registering a user

| | | | | |
|---|------|-----|------|------|
| https://patientapp-server.biotronik-homemonitoring.com/public/api/version | POST | 200 | 38b | 75ms |
| https://patientapp-iam.biotronik-homemonitoring.com/auth/realms/master/protocol/openid-connect/token | POST | 200 | 262b | 20ms |
| https://patientapp-server.biotronik-homemonitoring.com/public/api/version | POST | 200 | 38b | 74ms |
| https://patientapp-iam.biotronik-homemonitoring.com/auth/realms/master/protocol/openid-connect/userinfo | GET | 200 | 126b | 20ms |
| https://patientapp-iam.biotronik-homemonitoring.com/auth/realms/master/protocol/openid-connect/userinfo | GET | 200 | 126b | 19ms |
| https://patientapp-iam.biotronik-homemonitoring.com/auth/realms/master/protocol/openid-connect/userinfo | GET | 200 | 126b | 9ms |
| https://patientapp-data.biotronik-homemonitoring.com/cardiapp/_session | POST | 200 | 152b | 33ms |
| https://patientapp-iam.biotronik-homemonitoring.com/auth/realms/master/protocol/openid-connect/userinfo | GET | 200 | 126b | 16ms |
| https://patientapp-data.biotronik-homemonitoring.com/cardiapp/_session | POST | 200 | 152b | 23ms |

Figure 6.13: The requests in mitmproxy with 200 OK status responses

```

},
"Authentication": {
  "ServerUrl": "https://patientapp-iam.biotronik[REDACTED]/auth/realms/master/protocol/openid-connect/",
  "ClientId": "car[REDACTED]",
  "ClientSecret": "214[REDACTED]"
},

```

Figure 6.14: The ClientSecret and ClientId in strings.xml

itself, we also found that on the previous line of code was another credential, namely ClientId. This client secret is a randomly generated string of characters and numbers and appears to be in plaintext. The key is meant to be accessible only to the client and server. Further, it can be used as a secret key between the application and the authentication server to verify the application's identity [81]. It is generally recommended to keep the ClientSecret confidential and protect it from unauthorized access.

When looking at this information from *Configuration.json*, we wanted to see what kind of implications this could have for the security of the app. The OAuth Authorization framework includes registration of a client to an authorization server [34]. This document describes the authentication of a client to the server and states that the client can use the HTTP Basic authentication scheme. Alternatively, the client can authenticate by sending `client_id` and `client_secret` in the request body. These are the parameters that were discovered in the configuration file.

We revisited the requests sent to the server for login of a user, found in Figure 6.11. It was not possible to see any request being sent that authenticated the application itself (the client) before the patient credentials were sent (the end-user). However, it was observed that every request being sent to the servers included a Basic Authentication field in the header. All the requests had the exact same string for the authentication, as seen in Figure 6.15. After reading more about Basic authentication, we learned that this is a simple authentication scheme, that encodes a client id and secret key pair on the form "id:key" with Base64 encoding [8]. Base64 can easily be reversed by using online tools, such as `base64decode`⁶ or `base64` command-line tool. The results of this can be seen in Figure 6.16. The Basic Authentication string being sent with every request to the server is identical to the two strings found in the *Configuration.json* file, separated by a colon.

As a result, the two strings found in plaintext in the APK of the application can be used to authenticate the application with the server by using Basic Authentication. After encoding the strings with Base64, this authentication was used on all communication with the server. Consequently, as these strings were not encrypted in any

⁶<https://www.base64decode.org/>

| Request | | Response |
|---------|---|----------|
| Pretty | Raw | Hex |
| 1 | POST /auth/realms/master/protocol/openid-connect/token HTTP/1.1 | |
| 2 | Authorization: Basic Y2FyZ[REDACTED] | |
| 3 | Content-Type: application/x-www-form-urlencoded; charset=utf-8 | |
| 4 | Content-Length: 62 | |
| 5 | Host: patientapp-iam.biotronik[REDACTED] | |
| 6 | Connection: close | |
| 7 | | |

Figure 6.15: Header of the request going to the authorization server

Decode from Base64 format

Simply enter your data then push the decode button.

Y2FyZ[REDACTED]

i For encoded binaries (like images, documents, etc.) use the file upload form a little further down on this page.

UTF-8 Source character set.

Decode each line separately (useful for when you have multiple entries).

Live mode OFF Decodes in real-time as you type or paste (supports only the UTF-8 character set).

< DECODE > Decodes your data into the area below.

car[REDACTED]:214[REDACTED]

Figure 6.16: Decoding of a string with Base64 encoding

way, it would be easy and straightforward for an attacker to succeed in stealing this information. The attacker can take advantage of this vulnerability. By successfully obtaining these credentials, the attacker can impersonate the application to the authorization server. This means that an attacker can successfully get a phishing app authenticated, by using the credentials of the legitimate application. This gives unauthorized access to the data and makes it possible to perform unauthorized actions. Potential actions can be data theft, where an attacker can retrieve sensitive information stored on the server. Moreover, the attacker can manipulate the communication between the phishing app and the server, and perform an injection attack that can compromise the system. If the attacker is successful with a patient downloading the phishing app instead of the real one, it can steal the patient's credentials and use this to log into the real application. As a result, the attacker can potentially achieve all the sensitive information about the exposed patient in the original app.

For a more advanced attack, once the attacker obtains valid credentials from the user and logs into the original application, he can set up a proxy. Afterward, he can listen and analyze all the traffic to learn how valid requests to the server are sent. This can be utilized in the malicious phishing app connected to the same server, to send valid requests to the server and obtain data from other patients. Once a new patient downloads the phishing application, the attacker already knows how to send requests to the server and can mimic the behavior of the legitimate application. The attacker will be able to continuously gather information about the user without them raising suspicion.

6.2.6 Certificate Details

In section 6.1.3 we discussed the potential vulnerabilities for Medtronic when using weak algorithms in certificate signing. Despite the fact that this was not an issue given in the static analysis for Biotronik, we still wanted to print and analyze the certificate details in case of warnings the static analysis missed out. See Figure 6.17 for the result. As observed, Biotronik uses both SHA-1 and SHA-256, similar to Medtronic. Further, the app uses SHA-256 with RSA as a signature algorithm, in contrast to Medtronic which uses SHA-1 with RSA. Moreover, the key size of the subject public key algorithm is 4096-bit vs. 1024-bit used in Medtronic. Because of the 4096-bit RSA key, the certificate's signature algorithm is considered strongly secured and possesses no security risk as it is above the recommendation given by NIST. Lastly, no warnings appeared. To conclude, the Biotronik app appears to use secure and appropriate signature algorithms.

```

% keytool -printcert -jarfile PatientApp_2.6.0_apk-d1.com.apk
Signer #1:
Certificate #1:
Owner: CN=Android, OU=Android, O=Google Inc., L=Mountain View, ST=California, C=US
Issuer: CN=Android, OU=Android, O=Google Inc., L=Mountain View, ST=California, C=US
Serial number: 5cac6e52c1cd9692890941b4a3dbb3a5863a0a08
Valid from: Fri Aug 17 08:54:54 CEST 2018 until: Mon Aug 17 08:54:54 CEST 2048
Certificate fingerprints:
  SHA1: 92:CA:A2:C5:E6:D1:20:02:6D:AC:1F:B4:D3:B6:16:14:8F:CF:32:A7
  SHA256: 4A:13:64:D1:C3:03:04:8E:D0:4D:98:DC:E9:FB:87:7B:B5:58:CC:BA:D7:AB:C7:64:A9:68:3A:A1:54:C9:F9:74
Signature algorithm name: SHA256withRSA
Subject Public Key Algorithm: 4096-bit RSA key
Version: 3

Extensions:

#1: ObjectId: 2.5.29.19 Criticality=false
BasicConstraints:[
  CA:true
  PathLen: no limit
]

```

Figure 6.17: Certificate details of Biotronik

6.2.7 Unverified Issues

In order to maintain clarity, this section is dedicated to issues found during the static analysis, described in subsection 6.2.1, that were tested and analyzed. However, we could not prove that these issues are valid or have an impact on the security of the application. As a result, the remaining issues with a severity level of warning and high, which have not been previously discussed, are further investigated in the following sections and their outcome are presented.

Insecure Random Number Generator

The threat with ID 5 in Table 6.1 states that the Biotronik app uses an insecure Random Number Generator (RNG). We wanted to investigate if this issue can be exploited by an attacker, and eventually how. To check if Biotronik uses an insecure RNG, we used ADB (appendix A.3) and an emulator (appendix A.2). By using the command below, it displayed the logs related to the usage of the “java.security.SecureRandom” class in real-time.

```
1 $ ./adb logcat | grep SecureRandom
```

ADB is a powerful tool making it possible to communicate with the device. One feature ADB has is to monitor the system and app logs. If the application uses the insecure algorithm, it logs warnings and errors while playing around on the Biotronik app in the emulator. After attempting to bypass authentication and register a user, there was no sign of RNG.

Another way to check the presence of the “SecureRandom” class was to conduct code analysis. By searching through the code with this particular keyword, we noticed that the class was used in three different files, all under the folder “com.google.android.gms.internal.ads”. The code is using a library called

“java.security.SecureRandom” in Java, which is known to be cryptographically secure [69]. Therefore, we can say that the RNG was suitable for cryptographic purposes. As a result, we assumed that it could be a false positive in the static analysis done by MobSF. If we did not have any restrictions on accessing the application, we would conduct further investigation to confirm the use of RNG.

MD5

The vulnerability regarding the weak cryptographic algorithm MD5 is found in four different files in the application. They are all situated in the Google folder, within ads. All the files contain a method called getInstance(“MD5”). The only part of the code that is obfuscated, is the files placed in the “com.google.android.gms.internal.ads”. Functions containing getInstance(“MD5” are obfuscated, which made it hard to understand them. This again resulted in difficulty to understand what the algorithm was used for. It was not possible to determine the use cases where MD5 might be employed in the app. This could for example be verifying the integrity of files, generating checksums for data integrity checks, storing passwords, or data identification [90]. Even though we did not know what its functionality was, it was considered to be insecure and it would be a good security practice to change to a better hashing algorithm, such as SHA-256. However, it seemed as though there were only some functions of the analytics performed by Google that used this, and therefore the impact of this cryptographic algorithm might not be that severe.

Frida

After our login attempts described earlier had failed, we wanted to take a different approach to bypass the login functionality of the application. To execute this, we used Frida, as explained in section 4.3.3. This is a powerful tool that enables dynamic changing of the inserted values into functions while the application is running. We wanted to try to see in which order functions were called during the login process, in order to locate a function that we could alter the response to, in order to trick the application. Our hypothesis from earlier was that the connection to the Couchbase database was the last step and that we did not manage to fake this connection. We wanted to verify this hypothesis, by looking for the functions where we received the error earlier. As seen in Figure 6.8, we received an “replication auth error unable to get the sync gateway session token”, which was present in two different functions “DeprecatedPull” and “DeprecatedPush”. These functions can be searched for when running the application with Frida, to see if we can learn some more about what they ask for and how to bypass them.

In order to check for a specific function during the execution of Frida, we used the same command as described in the Medtronic analysis:

```
1 frida-trace -U -i "*DeprecatedPull*" -F
```

```

~ % frida-trace -U -i "*DeprecatedPull*" -F
Started tracing 0 functions. Press Ctrl+C to stop.
(^C%
~ % frida-trace -U -i "*DeprecatedPush*" -F
Started tracing 0 functions. Press Ctrl+C to stop.

```

Figure 6.18: The result from running Frida with the Biotronik app.

The `-U` is for an external device, the `-i` specifies the function to look for, and the `-F` states that it is on the frontmost application. However, when we ran this command the Frida tool stated “Started tracing 0 functions”, which means that it was unable to find this method. This can be seen in Figure 6.18. As we see from the figure, this was the case for both the Push and Pull of the aforementioned functions. Therefore, we tried to change the approach, in order to try and receive some information from Frida. Firstly, we tried to not specify any functions, just the application to run. This was not successful, as the program did not seem to understand what it was looking for. Thereafter, we tried to iterate through several different function names that could be associated with login, such as “authenticate”, “checkValidity” etc. None of these gave us any interesting findings either, even though the command itself worked with a more generic function name, such as “open”. After several attempts, we had to admit that we were not able to learn anything useful about the login process with the use of Frida. As there was not much time left to study this any further, we had to accept that we had to let this go. If we had more time, we would have explored the usage of Frida further, to try to use this in the bypassing of the login.

6.2.8 Summary of our Findings

In contrast to the Medtronic app, we had the possibility to do various analyses on the application which resulted in numerous findings, see Table 6.5 for the summary. The analysis of the mobile application’s network traffic in Burp Suite, code inspection, logcat in Android Studio, and mitmproxy revealed multiple important findings about the authentication mechanisms used and the server infrastructure in the application. Additionally, we achieved some unverified issues in subsection 6.2.7 that are not included in the table.

Table 6.5: The summary of findings for PatientApp

| Discovery | Explanation | Covered In |
|-------------------------------|--|------------|
| Geofencing API | The app uses background location for Geofencing, which keeps track of users' location in relation to places of interest. It is difficult to see the need for this feature. | 6.2.2 |
| Authentication protocol | The request sent to validate credentials indicated that the application used OIDC for authentication. This protocol is built on top of the OAuth 2.0 framework and provides Single Sign-On (SSO) capabilities which simplify the authentication process for users. | 6.2.3 |
| JWT | OIDC relies on JWT as the primary token format for authentication. Our attempt to forge a token was unsuccessful, which can indicate that the token verification is properly implemented and secure. | 6.2.3 |
| Backend server | The app uses the Couchbase Sync Gateway, which is based on the CouchDB database. It uses an older version of the gateway and has known issues. | 6.2.3 |
| Keycloak | The application use Keycloak as an IAM tool for authentication and authorization services. The interface was accessible through a URL. Keycloak is a part of the application's security architecture. | 6.2.3 |
| Server infrastructure | Four domain names were discovered. By querying DNS records and performing an IP WHOIS lookup, it was determined that the servers belong to Biotronik. Ownership information and details about the IP addresses were also obtained. | 6.2.3 |
| Password criteria | By tricking the mobile application into thinking we successfully registered a user, we obtained the password criteria for all users, which had a low-security level. | 6.2.4 |
| The ClientSecret and ClientId | Possible secrets strings shown in plaintext in the code. These were used for authenticating the app, using a simple authentication scheme that can be easily reversed. | 6.2.5 |

Chapter 7

Questionnaire

This chapter relates to research objective 3, section 1.4, “*Gathering information about the patient’s perspective by executing a survey and analyzing the results*”. The questionnaire is performed in order to give us a more accurate understanding of the patient’s stance and thoughts on the evolution of bringing a mobile application into the pacemaker ecosystem. All the questions in their entirety can be seen in appendix B. The questionnaire was posted in a closed Facebook group for pacemaker patients, named “*Young pacemaker patients*”. This group contains patients from all over the world.

With regard to the ethical concerns we discussed in section 4.5, the first page of the questionnaire included information for the respondents. We included information about the purpose of the questionnaire and its intended usage. We specified that the answers would be used in a master thesis, and gave contact information in order to give the respondents the possibility to reach us if they had any comments or questions. There was information present to explain that we did not collect any personally identifiable information and that the respondents had the possibility to skip any questions throughout the questionnaire if they wished. We also stated that the target groups were patients who currently use a mobile application connected to their pacemaker and patients that do not.

After the initial information page, the question pages arrived. We used a combination of multi-choice questions with single-select answers, Likert scale questions, and open-ended questions where the respondents could write themselves. For the Likert scale, we provided several statements, where the patient could respond how much they agreed with the statement from “Strongly disagree” to “Strongly agree”. The open-ended question was at the end of the questionnaire, after some additional information about why we asked these questions and the possibility to skip these questions if they wanted. The questionnaire was divided into different parts with their own overarching theme, and we splitted the presentation of results into the same sections.

7.1 Demographics and Clinical

In total, we received 38 responses to our questionnaire. We observed that not everyone has answered all the questions provided, which was expected as no questions were obligatory. For the demographics part of the questionnaire, we only asked the respondents their age and their gender. Both questions provided an opportunity for the respondents to answer “Prefer not to answer”. This option was not chosen on any of these two questions.

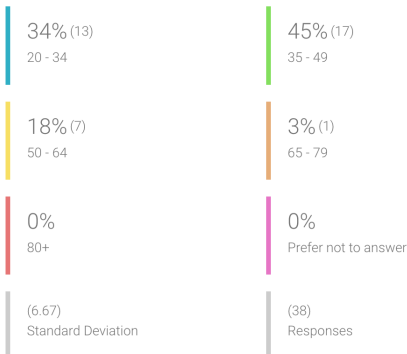
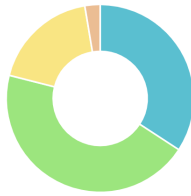
For the age demographic, it can be observed that the majority of the respondents, 45%, are in the age range 35-49. This equals 17 of the total respondents. Thereafter comes the 20-34 range, followed by the 50-64 range, and lastly the 65-80 range, with respectively 13 respondents (34%), 7 respondents (18%), and 1 respondent (3%). We did not get any respondents over the age of 80, which was our last option. The reason for this can be that the pacemaker group we sent out the questionnaire in was targeting younger pacemaker patients.

We also asked about the gender of the respondents, to get a better understanding of which group of people responded to the questionnaire. The result was that the vast majority of the respondents were female, at 86%, and 32 individuals. There were also 5% male respondents (2 individuals) and 8% non-binary (3 individuals). Even with the option of answering “Prefer not to answer”, we can observe that we had a total of 37 responses to this question, while there were 38 responses to the last question. The graph for both of the demographic questions can be seen in Figure 7.1.

After the demographic questions, we wanted to look at the clinical differences between our respondents. Therefore, we asked them both what kind of device type they currently have and from which manufacturer. When it comes to the device type, we included options for pacemakers, implantable defibrillators (ICD), and Cardiac Resynchronization Therapy (CRT). These are the three most common IMDs for patients with heart-related disorders. From the responses, we observe that 81% (31 individuals) have a pacemaker, while the number of ICDs and CRTs are 11% (4) and 8% (3) respectively.

The device manufacturer question was a bit more evenly distributed than the two previous questions. For this question, we included four common device manufacturers, and the option to answer “Other”. All the respondents to our questionnaire have one of the four mentioned vendors for their devices. The largest part of the group had Medtronic as their manufacturer, with 55% or 21 of the respondents. Tied for second was Abbot/St. Jude and Boston Scientific, with 18% of the respondents each, or 7 individuals. Lastly, there were 3 individuals who had Biotronik as their manufacturer, which totals 8% of the respondents.

1 Age



2 Gender

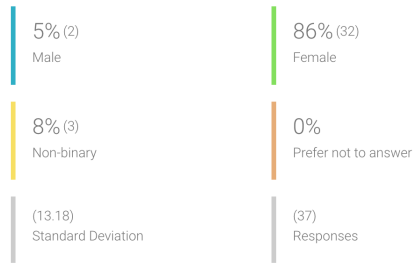
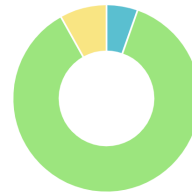


Figure 7.1: Graphs of Demographics

To conclude on the information gathered from these two sections, we observed that the respondents to our questionnaire are mainly females, between the age of 20 - 49. In addition, there was a large majority that had a pacemaker device as their ICD, and the manufacturer of the device was primarily Medtronic, Abbot/St. Jude or Boston Scientific.

7.2 Mobile Phone Usage

We included a section about the patients' mobile phone usage as we wanted to gain a better understanding of the respondents' habits when it comes to mobile phones. This included their habits on updates, making passwords, and general knowledge of password security. For this section, we used a combination of multi-answer questions and a Likert scale. The number of respondents per question is about 36 individuals.

The first question was regarding the type of smartphone that the respondents own. This statistic is interesting for us, as we only made a security analysis of Android versions of the applications. Therefore, it was interesting to see if the perceived security of the applications was correlated to which device the respondents own. We observed that the majority of our respondents had an Apple phone, with iOS. This amount constituted 67% or 24 individuals. On the other hand, there was 33% that

owned an Android phone. Once again, there was no one that chose to use the “Other” option for their response.

A Likert scale was used to get the respondents’ thoughts on five different statements in this part of the questionnaire. The scale ran from “Strongly disagree” to “Strongly agree”, where these represent 1 and 5, respectively. All the statements were written with personal pronouns, such as “I” and “me”, in order to make it easier for the respondents to reflect on their own behavior. The first three statements were related to the making of passwords, whether the respondents use different passwords for different logins, and have an active relationship to the security of their passwords. The two last questions were related to updates, where we asked both if the respondent update their phone and their application often.

The number of responses on the Likert scale questions was 35 for each statement, and the result is presented in Figure 7.2. The first two statements address whether the user has a strong, unique password and how advanced passwords the users make. Most are agreeing with these statements, where the first statement has a weighted average of 3.63 out of 5, where 5 again means “Strongly agree”. The second statement has a score of 3.77. This might be a result of password policies established for almost every password generation. Further, the third one states “I use different passwords for different logins in apps, websites, etc.”, where the participants answer to be agreeing as well, with a weighted average of 3.71 out of 5. When asked if they update the applications and their phones regularly, the participants answered closer to strongly agreeing with these statements. The statements achieved a weighted average of 4.2 on the fourth statement about updating apps, and 4.29 on updating the phone. Overall, the majority of the participants agreed with each of these statements, suggesting that they think about and have a concern for the security of their mobile phones. Alternatively, it could indicate that they may not be fully aware of what is good and bad mobile security practices. This is discussed further in section 8.4 about the difference between actual security and perceived security.

To gain further understanding we also asked multi-choice questions if the respondents update their application automatically on their phone, and how they make their passwords. The reason for this is that a Likert scale gives more of a comparative result, while multi-choice questions can give some more detailed and extensive information, to elaborate on the result from the Likert. For the question on application updates, 75% (27) of the respondents answered that they have turned on automatic updates. 19% (7) answered that they did not use automatic updates, while 6% (2) answered that they did not know. It is good security practice to update your applications whenever there is a new version published, as this might include bug fixes to issues that your device otherwise can be vulnerable to.

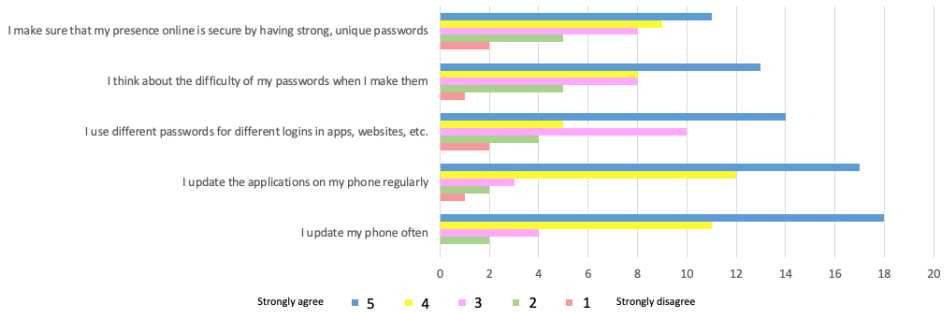


Figure 7.2: The result of the Likert scale questions on mobile phone usage

For the question regarding how the respondents make their passwords, we provided different alternatives of common password-making techniques. We wanted to see if the way the respondents make their passwords corresponds to their statement on whether they think about their password strength. The different alternatives we presented were to use random numbers and/or letters, memorable words/phrases, “suggested” from Google, Apple, etc., or not having a consistent method. There was also an option for answering “Not sure”. The results from this question are presented in Figure 7.3. As we can observe, the majority of the respondents use the “Memorable words/phrases” option for their passwords. This is not necessarily a very secure way to make passwords, as people often choose phrases that are related to themselves in some way. This can be the name of a child or pet, the birth year of someone in the family, or other similar words. When using such passwords, it could be possible for an attacker to quickly guess the correct password. This can be done by employing social engineering techniques on the victim or doing brute-force attacks with tools such as `psudohash`.¹ This answer does not correspond very well with the answer from the Likert scale, where a large amount of the respondents answered “Strongly agree” to the statement “I think about the difficulty of my passwords when I make them”, the second statement shown in Figure 7.2.

The last question of this section was regarding whether the respondent stores their password in a password manager. In addition to this question, we asked what type of password manager they use, for those who answered “Yes” in the multi-choice. In order to explain what we meant by a *password manager*, we included examples for the respondents. The examples were Google password manager, 1Password, and writing them down in a notebook. Of the 16 individuals that answered Yes to using a password manager (46%), 14 of these further answered which password manager they utilized. The password managers mentioned in this field were: Last Pass, OnePass,

¹<https://github.com/t3l3machus/psudohash>

12 How do you make your passwords?

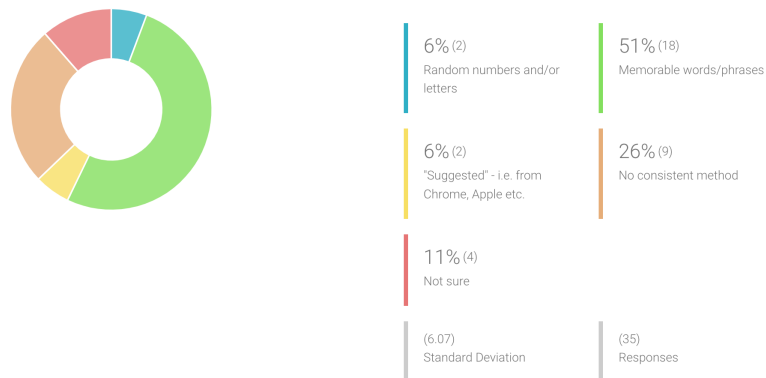


Figure 7.3: Graph of how the respondents make their passwords

Smartsheet, Apple Passwords, Keeper app, Proton, Google Password, and notes on their phones. Of these, Last Pass was mentioned the most. There were also two responses of “Don’t know”. On the question of whether they use a password manager, there were also a few that were unsure, 4 individuals, totaling 11%. Lastly, 43% of respondents, 15 individuals, answered no to password manager usage.

7.3 Perception of Applications Connected to IMDs and Cybersecurity

The next section considered the mobile application aspect, where we investigated patients’ perceptions and habits when it comes to mobile applications connected to IMDs. In addition, we asked some more general questions about the patient’s relationship to cybersecurity. This part contained one multiple-choice question, six Likert scale questions, and seven open-ended questions. The open-ended questions gave patients the opportunity to provide more details and allowed us to deliver a better discussion on this subject.

The first question asked the respondent if they use an application connected to their pacemaker. 23% (7) answered “Yes”, 74% (23) answered “No”, while one respondent answered “Not sure”, in total 31 respondents. An examination of the individual answers showed that 6 out of 7 participants that responded yes have the mobile application from Medtronic, and one from Boston Scientific.

The result from the Likert scale questions is presented in Figure 7.4, where each statement had around 30 respondents. The first statement was whether they use one or more apps to keep track of their health, and a large part of the respondents

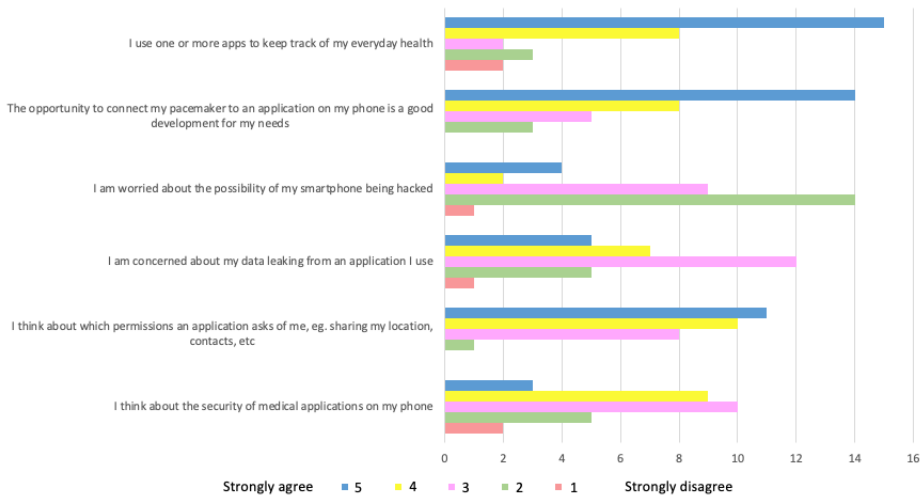


Figure 7.4: The result of the Likert scale questions on perception of application connected to IMDs and cybersecurity

agreed, with a weighted average of 4.03 out of 5. The next statement assessed the respondents’ perception of having an application connected to a pacemaker, stating that it is a good development for their needs. Similar to the previous statement, the majority agreed, with a score of 4.1. The rest of the statements were related to the cybersecurity aspect of the application and mobile phone. The third statement presented in the figure states that they are worried about the possibility of their phone being hacked. In contrast to previous statements, a larger portion of the respondents disagreed, with a weighted score of 2.8 out of 5. This can be a result of people having confidence in the security measures on their phone, where they trust the security of the device, they have low personal risk, and/or lack of awareness. Additionally, the fourth statement also regards security concerns, “I am concerned about my data leaking from an application I use”. Here, the participants have a weighted average of 3.33 out of 5, which indicates that the majority are neutral toward the statement. When it comes to the awareness of permissions, which is the fifth statement, the average score was 4.03. This means that they agreed on average, stating that they think about which permissions an application asks them. Lastly, we gave a general statement saying they think about the security of medical applications on their phones. The weighted average for this statement is 3.21, indicating a neutral value where the average participant neither strongly disagrees nor strongly agrees.

The last segment of the questionnaire contained seven open-ended questions regarding their thoughts and opinions on the application, its security, and general security aspects. The respondents’ engagement was good, with an average response

rate of 21.5 respondents. To visualize the answers, we utilized word clouds from a free online word cloud generator². This approach highlighted words and phrases that occur more frequently in the responses, giving them greater visibility in the word clouds.

The first two questions considered their thoughts on the benefits and drawbacks of an application connected to their pacemaker. The question we asked was “What do you think are the benefits of an application connected to your pacemaker?”, and the result is presented as a word cloud in Figure 7.5a. As seen, words and phrases that occur frequently are real-time, self-monitor, and ease of use. The patients are able to observe real-time happenings, which can provide them reassurance and/or gather information about their condition and their pacemaker performance. In addition, they are able to see events and trends as they happen, instead of twice a year at the device clinic. This can provide peace of mind as the patient have more knowledge of their status. For the monitoring part, the patients are able to constantly record and monitor their condition, allowing them to send reports of a cardiac event directly to their cardiologist. This can result in fewer appointments in person. Furthermore, the application is easy to use and convenient, for instance when out traveling. One respondent specifically mentioned that the application had reduced the number of in-clinic appointments by half, thereby lowering the risk of exposure to COVID-19, and saving time and money that would have been spent on appointments. In addition, the respondents liked that they no longer have to keep track of a separate device.

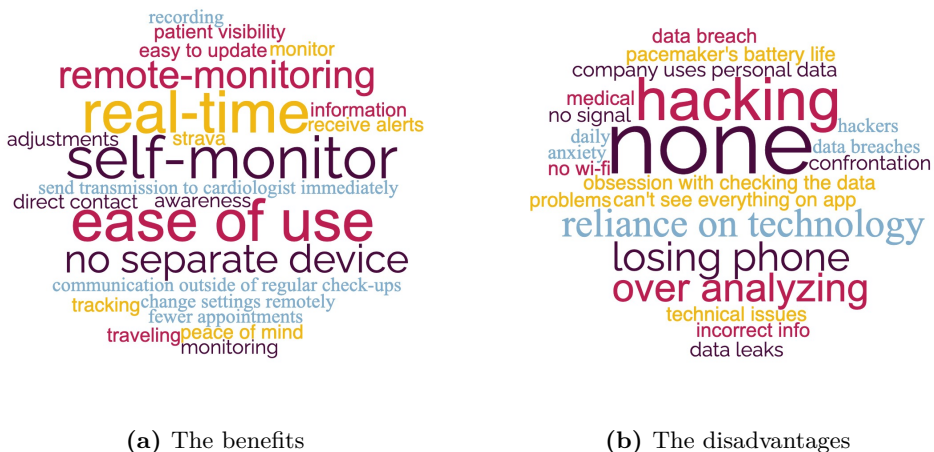


Figure 7.5: The respondents’ thoughts of an application connected to their pacemaker

²<https://www.wordclouds.com/>

The following question asked was “What do you think are the disadvantages of an application connected to your pacemaker?”. The word cloud of the drawbacks provided by the respondents is demonstrated in Figure 7.5b. While several disadvantages were mentioned, it is worth highlighting that the word that occurred most frequently was “none” when addressing the drawbacks. The reason for this might be a lack of knowledge and awareness, limited experience, or a positive overall experience of the application. Nonetheless, they also mentioned the potential of being hacked, losing their phone, and relying too much on technology. One participant emphasized the importance of patients being in tune with what their body is telling them, stating that “Numbers and statistics reported by a mobile application should always be verified by a professional.” Another respondent mentioned that their personal data being used by the company could be a disadvantage. However, they also noted that in today’s generation, where Apple and Google have access to lots of information about individual users, they don’t see how a medical technology company could be any worse in terms of data privacy and security. Another respondent underlined that despite the fact that so much data is collected, they are not allowed to see or access any of the data.

An interesting drawback mentioned of having an application connected is the possibility of over-analyzing the data and developing an obsession with constantly checking it. This can increase anxiety for some people, which contradicts the benefit of tracking to provide peace of mind. One respondent expressed their complete dependence on their pacemaker device for monitoring their heartbeats. They underlined concerns about connecting a mobile app and interrogating the device on a regular basis, as they believed it might have a negative impact on the pacemaker’s battery life. In addition, they were concerned about the possibility of unintentionally changing settings and thus preferred not to take the risk.

The third question was directed towards respondents who do not currently have the application, asking them, “What would make you consider using such an application?”. Their response is presented in Figure 7.6. We observe that the most commonly mentioned factors are the application’s ability to provide patients with access to their data and its functionalities. One of the respondents pointed out that the app provides better access, especially for travelers who prefer the convenience of an application instead of carrying a large transmitter box. Another respondent considered the possibility to access data and reports on daily status, pacing status, etc. Additionally, education about the pacemaker device is specifically mentioned as a significant consideration for using such an app. Lastly, there is one user who also mentioned that the user interface must be good, including accessible options, for them to consider the app.

After finding out their considerations of using such an app, we wanted to highlight



Figure 7.6: The respondents answer on what would make them consider an application connected to a pacemaker

one of the main functions of the app and asked the respondents “What are your thoughts about the opportunity to keep track of your pacemaker device and its data?”. Out of 24 answers, 19 of the respondents were positive and excited about the data tracking, and many of them would prefer to see more data in the app. One possible reason for this could be that today’s technology already offers wearables that collect health data, so the concept of tracking health information through technology is not unknown to them. In addition, the group of respondents for this survey is relatively young, and thus they are often more positive toward technology. It was highlighted that keeping track of data on the app could improve their ability to monitor their condition, reduce anxiety, improve their ability to cope with arrhythmias through pacemaker correction and body adjustments, and in the end give them peace of mind. On the other hand, some of the respondents expressed that they are worried and do not like the idea of having access to data that they can’t possibly interpret. Further, one respondent emphasized that they already had access to their pacemaker data for a long time, and did not find any data on the app useful. They mentioned that the data they have access to from their remote monitor covers their needs already and that there should be an opportunity to access real-time information about arrhythmia events. It was also mentioned that accessing what has been transmitted instead of the transmission status would be preferred.

To gain an overview of the respondents’ perspectives on mobile security in general, we asked the question: “What are your general thoughts on mobile security?”. Some

participants demonstrated an awareness of mobile security, acknowledged the need for improvement, and expressed concerns. On the other hand, there were also respondents who admitted not thinking much about mobile security or considering themselves naive about the topic. Some of the respondents admitted potential for improvement, without naming ways they could become better. Interestingly, some respondents said they were more worried about losing their phone than about security issues. The respondents' perspectives on mobile security were mixed, which were in line with our expectations.

After gaining a high-level overview of their thoughts on mobile security, we wanted to redirect to the medical device topic. Therefore, we asked them "Have you ever thought about the security of your medical device?" and the answers were split in half. The first half simply answered "No", with no further explanation. Some of the respondents admitted that their awareness was on and off, they did not think much about it except when it was on the news. Another respondent provided an honest response, stating that they did not understand why anyone would want to track them down as they considered themselves to be uninteresting. In contrast, the other part answered "Yes", and one respondent mentioned that they do not like to sign a document allowing Medtronic unlimited access to their device. Another one highlighted that they have done much research into how secure their device is. Further, one believed that the security of their device would never be compromised, as most of these cases happen in the USA, whereas they live in Europe.

Exploring further into the previous question, we finished the open-ended questions with "Are you aware of any previous security issues related to your medical device". 18 out of 22 answers expressed that they were not aware. One specified that they were aware that there have been Medtronic issues on older versions, but not on their specific device, but these have been resolved and mitigated for now. Lastly, one of the respondents had heard about issues regarding Bluetooth-compatible devices, but not theirs. Some were also very assertive in stating that there had been no issues related to their device.

An interesting aspect was examining the responses of patients who have the application to gain insights into their perceptions. Therefore, we investigated deeper into their answers. Most of them were satisfied with the application and its functionalities and said it benefits them in daily life, without raising any noteworthy concerns. However, one disadvantage of the app frequently mentioned was the limited amount of accessible data. Further, the majority did not think about the security of their medical device and none of them expressed any thoughts about mobile security in general.

In conclusion, the respondent's perceptions and habits regarding mobile applica-

tions connected to their IMDs were investigated. The majority of the respondents did not use such an application, but those who used it found the app beneficial. The security concerns varied among the respondents, with some of them expressing confidence in mobile security, and others worried about the possibility of data leaks and getting hacked. Overall, the participants were positive when it came to tracking their devices and their data, and very few were aware of previous security issues. The results from this chapter are used when we discuss actual security vs. perceived security from patients in section 8.4.

Chapter 8

Discussion

This chapter address and discuss the findings provided within chapters 5, 6, and 7. Based on the results from the security analysis and the questionnaire, we describe the vulnerabilities that arise and how they can potentially affect different groups of people. First, we discuss the findings from Medtronic and Biotronik’s systems and suggest improvements. Further, strengths and drawbacks related to the use of mobile applications were presented. The subsequent section compares the old hardware HMU with the new evolution of mobile applications in the pacemaker ecosystem to answer parts of our research question, as described in section 1.4. Next, from the questionnaire, we evaluate the actual security of the applications in contrast to the perceived security obtained from patients. Lastly, we provide some limitations that have affected our work.

8.1 Security of the Mobile Application System

The findings from the security analysis of the two separate mobile applications are taken from chapter 6. This is the section that answers our research question about potential cyber security concerns when it comes to patient safety and privacy on mobile applications connected to the pacemaker. Additionally, we provide suggestions for improving the security of the discussed mobile applications. We recommend adopting additional countermeasures based on previous security analyses to enhance the security of the applications.

8.1.1 Discussion on Findings from Medtronic’s System

When starting with the analysis of the Medtronic application, the first step executed was a static analysis of the application. To begin with, we only used MobSF for the analysis, which gave a low-security score to the application, 41 out of 100. This was a low score, and therefore, we thought we would find a lot of vulnerabilities in the application. Although the analysis stated where in the code the issues were situated, we later learned that there seemed to be some false positives within the issues of the

MobSF analysis. Therefore, we utilized an additional static analysis tool, to see if this provided the same weak result for Medtronic. When running the application through the BeVigil analysis, it received a security score of 8.7 out of 10 - or 87 out of 100. This implies more than doubling the score from one analysis tool to another.

The significant difference in results between the two tools sheds light on the fact that these static analysis tools are automatic, and therefore one should never take the results as a fact. It is important to be aware of the ability for error, even from these types of tools. In addition, there seemed to be different focus areas for these tools, which also can affect the score given to a specific application. This is also something to be aware of, as it is easy to focus solely on one analysis tool and thereby lose sight of the areas that this particular tool does not emphasize. A static analysis tool is an excellent basis for discovering possible issues within an application, but it is important to always look further into the findings from such a tool, as they might not always be accurate.

In Medtronic's case, for instance, we were surprised to see that the application received such a low score from MobSF. During our further analysis of the application, it was quite difficult for us to find any vulnerabilities. It was however mainly due to the powerful obfuscation of the application that we were not able to analyze the findings from the static analysis or other parts of the code base properly. Therefore, it is also challenging to make any statements regarding the security of the rest of the application. There is a possibility of the obfuscation technique being their main security mechanism, which would not be a good practice. For skilled attackers or attackers with appropriate resources, it is possible to deobfuscate the code base, no matter how well the obfuscation tool performs. There exist very powerful deobfuscation tools online. These are usually quite expensive, but for an attacker with some means available, it could be feasible to deobfuscate the code base of the application. Therefore, it is important to have a high level of security in the communication protocols, storage of data and so forth as well.

However, we were able to find that the application uses SHA-1 for the signature of the certificate. This is a weak hash function, that is no longer considered secure. It is vulnerable to hash collisions and has been proved to be practically broken, by researchers that were able to choose two different PDF files that resulted in the exact hash value. Therefore, SHA-1 is not advised to use in any signature functions. The use of SHA-1 in MyCareLink Heart was to some extent surprising, as the rest of the application seems to focus on keeping the security high. This seemed inconsistent with the security measures implemented with the obfuscation of the code base. It is a flaw that an attacker can make another certificate that receives the same signature hash, as this makes it possible to forge an application that is accepted as the Medtronic application.

Threat Modeling vs. Identified Threats

In chapter 5 we went through the ecosystem with Medtronic's application and found possible threats against the mobile phone and application, shown in Table 5.3. In addition, we found some threats against the communication between the components in the system, as seen in Table 5.6 and Table 5.7. We compare the identified threats in our threat modeling with the actual security issues identified in the analysis chapter 6. To keep the discussion concise, we compare only the threats which we have identified and the actual issues discovered during the analysis. Our focus is on the application and its communication, excluding threats against the mobile phone itself.

The first threat associated with the mobile application is ID 5, which is related to an attacker pretending to be an authorized application that the patient downloads. We did not find any indication that credentials were available in cleartext in the application during our analysis. However, there exists a possibility of forging an application and attempting to convince patients to use this app instead. By doing so, the attacker could gain the login credentials of a valid user, which can be used to access the actual application. Additionally, we found that the application's certificate is signed with a weak hash algorithm. In practice, this means it is possible to create another certificate that generates the same hash value for the signature. This could result in the acceptance of the forged application as the Medtronic application, as they share the same signature.

ID 6 concerns the attacker bypassing the authentication of the application if they have physical access to the device. If such an attack was executed, the attacker could gain the view of a logged-in user and observe their health data. However, it seems as though there are several tools that cannot be used with the application because of the checks implemented by the Arxan obfuscation. This makes a process like bypassing the authentication more troublesome. In such a situation, it seems most likely that the attacker is able to access the view of a logged-in user by guessing the user's password correctly. This is related to ID 9, which is about the attacker brute-forcing the password of the user. We were not able to verify if there was a limitation to the number of attempts for user logins. Therefore, this might not be possible to perform either.

The next threat is ID 7, which relates to the attacker's ability to hijack the application and send wrongful data to the doctor or show fake information to the user. This requires the attacker to gain unauthorized access to the application and then be able to change the information displayed to the user during the runtime of the app. The other option is to interfere with the information when it passes through the application and change the content before it is sent to the doctor. Both of these require an authenticated view of the application, which we have already concluded

might be difficult to achieve. There are also checks in place to prevent the tampering of the code during runtime in the services offered by Arxan for obfuscation.

For the threats concerning communication over the internet, we focus on ID 25, 26, and 28, as these are the ones we have tried to check the validity of. That is not to say that the other threats in Table 5.6 are not applicable, only that we do not have enough knowledge to comment on them. For Table 5.7, we focus on ID 31 and 32. Again, that is not to say that the rest of the threats are not applicable.

Firstly, we have IDs 25 and 26, concerning the interception and alteration of data on the interface between the phone and the server. According to the information on Medtronic's own websites, the communication between the phone and the backend network is over regular Wi-Fi [52]. We were unable to study this traffic directly but assumed that the communication is encrypted, seeing as the communication between the phone and pacemaker applies encryption. If that is the case, one way to perform an interception of the communication is by inserting a proxy between the phone and the server. This requires the phone to accept the certificate of the proxy, but once this step is completed, all communication can be intercepted. It is also possible to set up a fake base station between the phone and the server, but this does not necessarily allow the attacker to read the data being sent. For the ID 26 attack, it is not necessary to be able to understand the communication flowing between the two devices, although it is an advantage. As long as the attacker receives the information going from the phone to the server, they can alter the packets, which leads to incorrect information going to the server.

The next threat has ID 28 and detailed a DoS attack where the attacker blocks the transmission of data from the mobile phone, making the doctor unaware of the patient's status. This is a plausible attack, but would most likely require the attacker to block all communication from the mobile phone, such as by jamming the frequency the phone sends signals on. This would require a jamming device situated in appropriate proximity to the phone, which is impractical. Additionally, it would quickly be noticed by the user if all signals are gone. Therefore, it would be challenging to maintain this blockage for longer periods of time, which is necessary to effectively prevent the doctor from having sufficient information about the patient.

Moving on to IDs 31 and 32, concerning communication between the pacemaker and the mobile phone. These are similar to the ones already discussed, where one concerns the interception of data between the pacemaker and phone, and the other involves tampering with this data. When it comes to the communication between the pacemaker and phone, it is harder for an attacker to insert themselves as a MITM. This is because the communication was over BLE, and thus no base station or other components are in between the two communicating parts. In addition, the proxies we

have discussed in this thesis are Hypertext Transfer Protocol (HTTP) proxies, which do not work for this communication. However, there exist tools that can be used to sniff BLE communication. Even if an attacker is successful in doing this, they need to understand how BlueSync works in order to make sense of the data from the pacemaker. Therefore, it is uncertain to what extent such an attack can be executed.

Improvements

We begin by introducing a best practice for Medtronic based on previous security. Thereafter, we offer two suggestions for potential improvements, focusing on areas Medtronic can enhance its security. It is important to note that both suggestions may not be directly applicable, but can be considered.

As discussed in subsection 8.1.1, the application uses SHA-1 as the signature algorithm for the certificate. Naturally, the best practice of security is to upgrade the hash function, such as SHA-256 or SHA-512, to use a stronger algorithm. Additionally, using a strong hash function with a strong key length is also recommended best practice to mitigate the risk of an attacker breaking the encryption. A key length of 2048 bits or more is considered strong, and thus we would advise Medtronic to increase the key length from the 1024 bits they currently use to follow best practices.

One suggestion we have for Medtronic is to establish a process for regular updates on security to stay up-to-date and to discover known vulnerabilities of their mobile application. Discovering a vulnerability as early as possible saves them time and money when updating the mobile app to secure the patient's privacy and safety. In general, improving the certificate management process by using strong cryptographic certificates and following the industry's best practices is suggested. Medtronic should conduct regular reviews and update certificates to prevent the use of weak or compromised certificates. Having a thorough process for security updates can help prevent the exploitation of newly discovered vulnerabilities.

Lastly, we suggest increasing the number of supported devices of Android phones. We understand choosing to start with only one type of device, especially when the security requirements need to be high, to ensure a secure application for the users. However, if the number of devices supported by the application increases, this would improve the ease of use for the patients. As it is now, a large number of patients would have to switch out their phones in order to use the application. It should be a goal for Medtronic to be accessible to the majority of the patient group, and thus they should look into what can be done to increase the number of supported devices.

8.1.2 Discussion on Findings from Biotronik's System

When executing the static analysis for Biotronik's application, we observed the same findings as we did for Medtronic. The application received a score of 50 out of 100 from MobSF, and while this is better than Medtronic, it is still a mediocre score. When running the same APK file through the BeVigil analyzer, the application received a 9.1 out of 10, or 91 out of 100. This was a significantly higher result. We observed that for Biotronik's application as well, the BeVigil analysis gave a score that was a lot closer to excellent than what MobSF provided. This once again highlighted the importance of double-checking the findings from such automated tools, and not blindly accepting them as true. During our deep dive into the issues from the static analysis we found several results that turned out to not be any vulnerability in the application in practice.

We can also note that Biotronik's application generally received a higher score from both the static analysis tools than Medtronic's application obtained. The difference was not significant, but we would expect the Biotronik application to be a little better at security measures. At first glance, the Biotronik application was more transparent, as the code base was not obfuscated. The classes had more descriptive names, and it was easier for us to search for important keywords in the code. In addition, there were no checks applied to see whether the application was run in an unsafe environment, such as on an emulator or on a rooted phone. This allowed us to open the application on our emulator, leading to many more possibilities to explore whether we could exploit the vulnerabilities. In this regard, the Biotronik application had fewer security mechanisms present, which in turn broadened our attack surface for the application.

During our ethical hacking process, we had the opportunity to access and explore the application within the emulator. To further test the application, we set up a proxy on the emulator. However, it is important to note that no actual interaction with the server was conducted. By looking at the messages being sent to the server we were able to gain significant knowledge about how the application was structured. We learned about the different servers Biotronik uses and the flow of the authentication of a user trying to log in. This, in turn, was utilized to try and bypass the authentication for the application, register a new user, and log in as a fake user. We were able to fake the correct responses from the server to complete the registration process for a new user. This allowed us to see the password criteria for the users of the application, as well as gain information about which information the user must provide to register their device. The difficulty of adding a proxy to the emulator was quite low, which in turn helped us to gain a lot of understanding about the inner workings of the application. This is not good for the security of the application, and thus the addition of a proxy should be harder to perform.

With regard to the authorization of the application, we were able to locate the credentials that the app uses to authenticate with the server. Both the ID of the client and the secret were located in cleartext in a file in the code base. We also observed that all requests going from the application to the server were authorized by a Basic authentication scheme. This scheme uses Base64 encoding, which is easily reversed by using an online tool. When reversing, we could see that this authentication matched the two strings that we had found in the aforementioned file. This was one of our most important findings from the analysis, as the presence of these strings in cleartext implies the possibility of an attacker impersonating the application to the server. We have already presented the possible implications of this finding in subsection 6.2.5. These attacks are feasible to perform and it is not unlikely the patients can be influenced to download another application than the one presented in the information from the hospital. It is possible to have a similar icon and name so it is possible to mix them up, or an attacker could send download instructions as mail to the pacemaker patients with a link to the fake app. However, Google Play Store and Apple Store have robust security mechanisms to mitigate the risk of phishing apps being available for download.

As we were unable to successfully log in to the application, there are several vulnerabilities that were not fully explored. Given more available time, it would have been beneficial to further investigate these issues in greater depth. In the login process, we suspect that the step we were unable to bypass was the authentication to the CouchBase server. In addition, our JWT was not valid. These are positive findings for the security of the application, as it implies that there exist some checks for validity. However, as we are not able to log in to the application, we are also unable to make any statements about the security of the internal functions of the application. As an example, we wanted to look at the storage of data the application receives from the data server, to see if this is done in a satisfactory way.

Threat Modeling vs. Identified Threats

In chapter 5 the analysis of the ecosystem involving Biotronik's application revealed potential threats to the mobile phone and the application itself. These threats can be seen in Table 5.3. Furthermore, we identified threats against the communication between the different components of the system, seen in Table 5.7. In this section, we compare the possible threats identified in the threat modeling and the actual issues discovered in the security analysis in chapter 6. We limit our discussion by comparing only the threats which we have found in threat modelling and the actual issues from the security analysis.

The ID 5 of the table concerns the possibility of an attacker pretending to be an authorized application that the patient downloads. Then, the attacker gets

hold of the patient's data. This is a threat observed to be possible throughout the analysis, as we were able to find the credentials of the Biotronik application in the configuration files. However, as stated in the threat modeling, this attack has a low impact, as the potential attack can affect a small group of patients' privacy, but is not harming their safety. Secondly, we have ID 6, regarding the attacker's ability to bypass authentication with physical access to the device. This particular threat required significant effort and time to validate, but unfortunately, we were unable to successfully demonstrate its validity. Nevertheless, this is not to say that this threat does not exist for the Biotronik application. It could still be possible to bypass the authentication. Still, it requires time and comprehensive knowledge of the different parts of the system that is in play for the process of authenticating a user.

Further, we have ID 9, related to an attacker brute-forcing the password of a user to gain access to a logged-in view. For this to work, the attacker requires physical access to the device, as well as a valid email belonging to a user of the application. We have not been able to find any limit to the number of logins a user can test, and thus this seems to be an attack that is plausible. Combining the lack of restrictions on attempts with the weak password requirements we found in the registration of a user process (section 6.2.4), this can potentially be an attack that does not require too much computing power or time for an attacker either.

For ID 13, related to an attacker's ability to gain access to other parts of the phone because of app permissions, we identified some findings. The Biotronik application asks for permission to read and write to the external storage of the device. Therefore, if an attacker can hijack the application, he will be able to read the storage of the mobile phone. This also makes it possible for an attacker to add things to storage, and overwrite previously saved items. In addition, the application asks for camera permission. This makes it possible for an attacker to access the camera whenever the application is in the foreground. Both of these permissions allow an attacker to access other parts of the mobile phone of the user, without them being aware.

Moving on to the communication between the phone and the backend server, we focused on ID 25 and 26 from Table 5.6. As mentioned earlier, this does not mean that the other threats present in the table are irrelevant. Our focus has primarily been on specific threats for which we have gathered a significant amount of information. We did not discuss ID 28, which we did for Medtronic, as there was little communication of importance moving from the phone to the server, and there was no real danger with the information being blocked.

ID 25 and ID 26 are connected as they both concern intercepting the communication between the phone and server, with one being just a passive listening attack and the other tampering with the data being sent. For the communication between the

phone and the server, we have observed that HTTPS with Transport Layer Security (TLS) is used. This means that the communication itself is encrypted, and hard to read for an MITM. However, if an attacker is able to set up a proxy, as we have done in the analysis, the communication flowing between phones and servers is easy to get ahold of. Another option would be to set up a fake base station between the two communicating parties. However, this would not automatically make the attacker able to decrypt the communication, even though he would be able to change the packets flowing through. To intercept the traffic and make sense of the content, the most efficient approach would be to install a proxy that listens to the communication. In order to read the HTTPS communication, we need a certificate installed on the phone of the patient.

Improvements

We present best practices and options for the Biotronik app based on our results. Further, we provide suggestions for improvements if we were to advise Biotronik of what can be done better. Naturally, all the suggestions may not be applicable to the mobile application.

Biotronik needs extra secure practices for storing sensitive information, such as client IDs and secrets. A best practice would be encrypting the sensitive information, or the entire JSON file, and implementing access control for authorized users to have access. In addition, it is important to secure the storage of sensitive information which includes servers, intrusion-detecting systems, firewalls, etc. Even though the attacker is able to obtain the file, encryption ensures that the content is unreadable and unusable without the correct decryption key. This requires that the key is stored securely, for example by using Android Keystore.¹

Another best practice is to train patients and make them aware of how to prevent and mitigate user risk, which reduces the threat of being exposed to a phishing attack. Being aware of social engineering and phishing is also presented as an important best practice in [46]. Biotronik already provides installation and registration guidance. If this is educated to patients properly, it helps to address the potential risk of users being influenced to download malicious apps from a third-party website or unknown sources. Unfortunately, the password criteria presented in Figure 6.12a does not contribute to the importance of the patients having strong, unique passwords on the app.

The best practice when it comes to the login process is to automatically log out the user after a time period, to protect the health data in the app. The Biotronik application has the possibility to choose “Remember me”, which we assume keeps the

¹Android Keystore: <https://developer.android.com/training/articles/keystore>

patient logged in. Another security recommendation is to remove this opportunity, which makes it harder to read the data even though an attacker has physical access to the phone. For better user-friendliness, it could be advisable to implement a biometric login for the patients. Furthermore, if the application cannot support biometrics, it should implement a Two-Factor Authentication instead. This is a second form for verification, to add an extra layer of security for best security practices. It is generally important to have user authentication as a high priority [46].

The first suggestion considered the underlying reason for the results obtained from the code analysis. To increase the difficulty of reading and analyzing the code for Biotronik, it was suggested to obfuscate the code, similar to Medtronic. Implementing code obfuscation could add an additional layer of protection against potential reverse engineering attempts and unauthorized analysis of the code. However, it is important to acknowledge that no obfuscation technique can guarantee secure protection. While code obfuscation can make it more challenging for attackers to understand and manipulate the code, it is not foolproof and other security measures should be included to ensure comprehensive protection. Furthermore, obfuscation increases the complexity of the code but this can make it harder for developers to perform debugging and troubleshooting. Therefore, it is important to think about all these aspects before implementing such protection.

The Biotronik app already uses a secure communication protocol, HTTPS with TLS. However, it is still possible to execute a MITM attack, where an attacker sets up a proxy, mimics the server's certificate, and intercepts and modifies the network traffic. Therefore, to enhance security, additional countermeasures can be considered. We suggest that certificate pinning should be implemented within the app. By enforcing this, only specific certificates are trusted in the application. This prevents interception by proxy servers with unauthorized certificates [15]. Without the possibility to add a trusted certificate to the mobile device, it reduces the ability to perform a MITM attack.

Couchbase introduces a concept called Field-Level Encryption (FLE) [28]. With this, Biotronik can encrypt sensitive fields and protect unauthorized users from gaining access to the database. FLE would mitigate MITM to some extent, as it is primarily designed to protect data at rest and in transit. It does not prevent all data leaks, for instance, statistics can still leak. However, it is important to note that it depends on which version of Couchbase is used. In the case of Biotronik, they would need an update to a compatible version (Couchbase version 3.0.5) to employ the concept, since as of now they have 3.0.4. Whether or not this is applicable for Biotronik, depends on what the data is used for. Unfortunately, we cannot determine what the purpose is since we were unable to successfully authenticate and explore more of the traffic on the app.

8.2 Benefits and Drawbacks of the Mobile Applications

In this following section, we look further into what can be the advantages and drawbacks of switching from HMU to a mobile application as the communication gateway in the ecosystem. In general, there are clear practical advantages to switching from an embedded device to a mobile application. The development of modern technology has led to a lifestyle where mobile phones have become an integral part of our daily routines. Consequently, patients nowadays can bring the monitoring device everywhere, instead of depending on staying home to access the communication gateway with the HMU. There exist additional positive aspects, but we divided the discussion into two parts. We discuss both the positive and negative aspects of having your device with you at all times and include some of the results from our questionnaire in the discussion.

8.2.1 Positive Aspects of Changing to an Application

The amount of people owning a smartphone continues to increase on a global basis. In this kind of environment, it is common to keep your phone with you at all times, as we rely on the device for a lot of functions in our everyday lives. Because of this established dependence, it could be a huge advantage for pacemaker patients to have their communication gateway as a mobile application on their phone. This is a device that they already bring with them whenever they are traveling, a device they are familiar with, and a device that they already use for different purposes. It can therefore be easier for some parts of the user group to learn how to use an application to transmit data to their doctor than to learn a new external device. An app can also be easier to use in everyday life for the patients, as it is not an external device that needs its own power supply and so forth.

One of the things that were mentioned the most as an advantage of using the application from the respondents of our questionnaire, was the ability to keep the communication gateway with them at all times. For some of the HMU versions, it is not possible to take the device along when traveling. This can be a large inconvenience for patients that travel for work or otherwise, as the data from their devices can not be sent to the hospital before they arrive back home. If anything were to happen with the device during the travel, the doctors would not be alerted. When an application is used as a communication gateway, the patient is not limited to transmission in one specific location. There is also a possibility for the patient to check on the go whether the transmission was successfully sent to the hospital, which can give peace of mind. Even though the possibility to check the status of transmission also exists on the HMU, this requires the patient to remember to look at it while they are at home. In addition, if something is wrong and the transmission has not been sent, the patient needs to stay situated at a close distance to the HMU

during retransmission. For the mobile application, retransmission can also take place while the patient is out, which improves ease of use.

Another huge advantage of converting the communication gateway to a mobile application is that it is cheaper and easier for the vendors. Making a dedicated hardware component for every patient that owns a pacemaker is costly. They need the materials to build such a device, in addition to software developers to configure the device correctly. The added cost for each new version of the HMU is high. If they can successfully use a mobile application as the communication gateway instead, this drastically reduces the costs associated with the communication between pacemakers and hospitals. The need for specialized hardware is completely gone, and the vendors only need to spend money on software developers to make and maintain the application. It is also easier for the vendors to continuously develop and update the gateway, as they only need to deploy an update of the application. This can lead to a more constant development process and also increases their ability to respond to found security issues.

8.2.2 Negative Aspects of Changing to an Application

Even though the benefits of moving the communication gateway to a mobile phone are present, there exist some disadvantages as well. First, it introduces new attack vectors. If you have a dedicated gateway that is situated in the patient's home, there is plenty of attacks one can perform on this system. Despite the previous theses on this subject have found a lot of vulnerabilities of the HMU and its communication channels, these are attacks that require skilled attackers with good resources and time. If the gateway is part of a mobile phone there are more potential attacks present. Not only can the attacker target the application directly, but it can also try to gain access to the patient's data through the surroundings. An attacker could intercept the communication from the phone, for example by gaining access to the network the device is connected to. In addition, they could access the mobile phone through another application on the device, and try to gain access to the patient's data through the common folders on the phone. The available attack surface increases, which also gives more possibility for attackers to get creative with how to access the data.

One can also argue that the population today is already too reliant on their mobile phones. If the phone gets lost or stolen, there are a lot of critical things that a person can lose access to, such as their banking services, contact info, or media and files with sentimental value. It is not necessarily smart to add sensitive health data to that list. In addition, if a patient were to lose their device, there could be several serious consequences. Theft of their data is one alternative, but this requires that someone steals their device and is able to open their phone and applications. A more

important consequence is that the patient stands without a way to send their device data from the pacemaker to the doctor if they lose their device. This is in most cases not necessarily a major problem, as they can log in to the application from another device that they buy or borrow, but might be a bigger issue if the patient is in a remote location without access to spare devices. The patient is totally dependent on the phone for communication with their doctor. In addition, the patient needs to keep Bluetooth on at all times, as this is the communication protocol used for the data being sent between the pacemaker and the phone. This can drain the battery life of the patient's mobile phone.

A concern from the respondents from our questionnaire was also that the increased availability of their own health data could lead to increased anxiety for some patients. The patients could become obsessed with checking their data at all times, and could potentially start to overanalyze their information. This could lead to a higher concern for own well-being, and perhaps even an increased number of calls to the clinics. The patient is also able to get a more active relationship with their pacemaker transmissions, as they are notified when transmissions are performed, or if there are any connectivity issues, and so on. Even though these notifications also existed for the hardware HMU, the patient could keep more of a balanced distance from it when the device was situated at home. With these reminders constantly present, the patient might become more aware of their own health situation. This can be negative, as they might feel inhibited from their heart condition by being constantly reminded of its existence.

8.3 Hardware HMU vs. Mobile Applications

We compared the hardware HMU with the result of the security analysis of mobile applications from chapter 6. To achieve the comparison, previous work on Biotronik's HMU is included. We have chosen to base the analysis of the HMU on a paper that works as a case study summarizing the previous theses written on the security of the HMU. The paper was written by Bour et al. and can be found at [12]. In this paper, they went through the findings on all the different versions of the HMU that have been tested by the SINTEF research group, with a focus on hardware, firmware, communication, and infrastructure vulnerabilities. Our results from the security analysis constitute the comparison of the systems regarding patient safety and privacy.

8.3.1 Security of the HMU System

We briefly state the findings discussed in the paper, in order to give a broad understanding of the tests that have been performed on the HMU and their outcomes. Firstly, for the hardware part of the analysis, they found that debug interfaces were

available for all the different versions of the HMU. This means that the authors were able to interact with the interfaces, which gave them access to dump the memory as well as the Flash, which again gave access to the firmware of the device [12].

For the analysis of the firmware, there were several findings. Firstly, they found that all the data of the device was stored unencrypted on the external Flash. This means that if an attacker has physical access to the device, they can dump the Flash and obtain all the data in cleartext. Secondly, they found that the firmware was not obfuscated or protected by encryption in any way. In addition, they found that log strings written in clear text on the firmware actually made the reverse engineering process easier. The memory was not protected either, which means that anyone with physical access to the HMU could copy it and gain access to the patient data if any is present. Next, they found hardcoded credentials and cryptographic keys, which the device used to connect to the network and backend server. These were unique per device but stayed the same for each connection attempt. The keys found were Advanced Encryption Standard (AES) and Data Encryption Standard (DES) keys, which ties to the found encryption algorithms. The proprietary protocol for patient data uses AES CBC, while single DES is used for the log data over SMS. DES is a broken algorithm from a security perspective, and an attacker can therefore get the log data by setting up a fake base station. The patient data can also be obtained by finding the AES key on the HMU and thereafter obtaining the patient data during transmission [12].

For issues associated with communication, the authors made several findings. Firstly, they observed that the credentials were sent in clear text to the modem from the microcontroller. This allowed them to get access to credentials used to connect to the manufacturer's Virtual Private Mobile Network (VPMN). They also found that there was no mutual authentication in place, which made it possible to spoof the backend server and trick the HMU to send the data to the fake server. Thirdly, the paper states that the two newest versions of the HMU use a proprietary protocol on top of TCP. This protocol was reverse-engineered by exploiting the hardware vulnerabilities found, in combination with the raw network data obtained by interacting with the device. Next, they observed that credentials used to authenticate to the backend server and VPMN were reused, and sent unencrypted in both cases. They also found that the communication between the pacemaker and HMU was unencrypted, which means that an attacker can read the data being sent if they are able to intercept the radio signal from the pacemaker. Lastly, they were able to perform a DoS attack against the modem of the HMU, which prevented it from communicating with the backend server before being rebooted [12].

From these findings, the report introduces three broader attacks that would be possible to perform. These attacks utilize the vulnerabilities found in the analysis.

First, they introduce a MITM attack that is possible to execute as they can spoof the identity of both the HMU and the backend server, as these are sent in cleartext. The second attack is unauthorized access to the backend server, which is possible by obtaining credentials from the HMU and accessing the VPMN. This would give access to the backend server, which could lead to a data leak if this server was compromised. The last attack was a large-scale DoS attack, as they found a way to make the HMU crash by sending a special SMS to it. In order to make the attack large-scale, the attacker would need to obtain the phone number of the devices, which could come from a leak from the manufacturers or from an internal source.

8.3.2 Comparison of the Systems

We perform a comparison of the different systems available in the pacemaker ecosystem today. There are currently three different versions available for the patient, depending on their pacemaker vendor. Firstly, the ecosystem where the HMU is the primary communication gateway between the pacemaker and the backend servers, without a mobile application available. Thereafter, there is the system with a HMU as the gateway and an additional mobile application to give patients more insight into their own data. Lastly, a version of the mobile application substituting the HMU completely. We did not perform a comparison of the two versions that include a mobile application against each other. We wanted to see the difference in security levels when moving from a dedicated hardware component to a mobile application, and also the security level of having the mobile application as an additional component.

It is important to note that the analysis of the HMU that we base the discussion on, was performed on different versions of Biotronik's HMUs. This means that we have no precise knowledge of how the HMU for Medtronic works. The vulnerabilities identified from the analysis performed in the paper [12] are therefore not necessarily applicable to the Medtronic HMU. One can say that making a proper comparison of the security differences from moving from the hardware component to the mobile application should be done based on devices from the same manufacturer, in order to make the comparison as accurate as possible. However, we did not have the time or resources to perform an analysis of any hardware HMUs produced by Medtronic. In addition, there were no devices from this manufacturer available in the SINTEF lab. Therefore, the comparison between the mobile application from Medtronic and the hardware HMU from Biotronik was carried out, despite not being ideal.

Comparison HMU vs. Mobile Application Substitute

As we can see from the Table 8.1, there exist similarities and differences between the two versions of the systems. We can observe that the Medtronic application seems to score better on security than the Biotronik HMU on some issues, such as obfuscation (ID 1), communication (ID 4), and cost (ID 7). We assume that the encryption of

Table 8.1: Comparison between the HMU and the mobile application from Medtronic

| ID | Factor | HMU | Medtronic app |
|----|----------------------|--|--|
| 1 | Obfuscation | Code not obfuscated | Code is obfuscated |
| 2 | Encryption algorithm | Uses AES-CBC and DES for encryption of data from HMU to backend, these are weak | Do not know which algorithm is used for encryption of data (BlueSync) |
| 3 | Signature algorithm | Do not know | Uses SHA-1 for certificate signing |
| 4 | Communication | Communication between pacemaker and HMU is not encrypted | Communication between server and mobile app is encrypted |
| 5 | Prerequisites | The user needs a HMU, internet, (for some versions) a continuous power supply | The user needs a Samsung or Apple device, new versions of the mobile operative system, internet, Bluetooth always on |
| 6 | Competence | Plug and play, set up by the hospital | Technical competence for registering a user and maintaining (e.g. updates) the app |
| 7 | Costs | Hardware, development, distribution of the HMU | Development of mobile app |
| 8 | Transmission options | Possible to send data to the doctor when at home, or when carrying the dedicated device (for newest version) | Possible to send data to doctor whenever and wherever |

the communication for the application is to be better than the one for the HMU, as the HMU uses weak algorithms. Another benefit of the Medtronic application is that it allows the patient to make transmission of pacemaker data to the doctor without being in their own home (ID 8). For the majority of the factors present in this table, we can observe that the application performs better than the HMU. However, the app uses a weak signature algorithm, but it is not relevant to compare this with the HMU as we do not know if it has any certificates needed to sign at all.

On the other hand, we can observe that the application has stricter requirements when it comes to prerequisites (ID 5) and competence from the user (ID 6). This is negative as the pacemaker is a medical necessity for a lot of patients, and therefore there should not be high demands for their equipment and skills in order to make

their device function as optimally as possible. According to a study performed by a Japanese hospital on the initial pacemaker implantation age of their patients, the age for implantation has risen in the last decades [48]. They found that for patients that received their initial pacemaker implantation during the 2010s, the average age was 75.8 ± 10 . In addition, the number of patients over the age of 90 at the time of first insertion was 5.2%. It should be noted that Japan has a population with a high life expectancy, but these results still give a picture of the evolution of pacemaker patients over the last decades. As the age of pacemaker patients increase, it might not be wise to advance the technology connected to their pacemaker too much, as they may struggle to keep up with the evolution. Particularly for Medtronic's application, which requires very specific versions of which mobile devices are supported, this might be a limitation for the patients. Both as it is an added cost for the patient if they need to change their device to a supported one, and as an added challenge if they need to learn a new, unfamiliar device.

Comparison HMU With and Without Mobile Application

The comparison between the pure HMU and the combined HMU and mobile application from Biotronik reveals both similarities and differences, as summarized in Table 8.2. As shown, there exist several contrasts such as the use of weak and strong encryption algorithms, where the mobile app uses stronger algorithms (ID 10). Additionally, Biotronik's mobile app incorporates encrypted communication with secure protocols, whereas the HMU does not encrypt (ID 12). It is also not relevant to compare the signature algorithm, but it is important to notice that the app from Biotronik uses a strong algorithm for certificate signature. As a result of these factors, we can determine that the mobile app is the most secure device in the system, even though the patient has the vulnerable features of the HMU as well.

However, if the patient only has the HMU, there is a lower requirement for prerequisites (ID 13), competence (ID 14), and costs (ID 15). As mentioned earlier, a trend in a paper from 2022 concludes that the average age at first implantation has increased by the years [48]. This trend may influence the use and requirements of technical devices, as older individuals may have less familiarity with them. Consequently, the level of technical competence among users could potentially impact the overall security of the system. If the patient group is increasingly older individuals, it might not be beneficial to introduce a system with two devices. Despite this, on Biotronik's own website², users express that they like that they can record symptoms and get details about their device on their smartphone when traveling. Considering these factors, it appears that there is a trade-off between providing users with more details and information and ensuring security. When more information is accessible

²<https://www.biotronik.com/en-de/patients/home-monitoring/patientapp>

Table 8.2: Comparison between the HMU and the combination of HMU and mobile application from Biotronik

| ID | Factor | HMU | Biotronik app and HMU |
|-----------|----------------------|---|---|
| 8 | Sensitive fields | Credentials in plaintext | Credentials in plaintext |
| 9 | Obfuscation | Code not obfuscated | Code not obfuscated |
| 10 | Encryption algorithm | Uses AES-CBC and DES for encryption | Uses HTTPS with TLS for communication and no further encryption in the mobile app. Uses AES-CBC and DES for encryption in HMU |
| 11 | Signature algorithm | Do not know | Uses SHA-256 with RSA algorithm for certificate signature of the app |
| 12 | Communication | Communication between pacemaker and HMU is not encrypted | Communication between server and mobile app is protected by HTTPS and TLS, while pacemaker and HMU is not encrypted |
| 13 | Prerequisites | The user needs a HMU, internet, (for some versions) continuous power supply | The user needs a HMU, internet, (for some versions) continuous power supply, and additionally a functioning smartphone |
| 14 | Competence | Plug and play, set up by the hospital | Technical competence for registering a user and maintaining (e.g. updates) the app, plug and play for the HMU |
| 15 | Costs | Hardware, development, distribution of the HMU | Hardware, development, distribution of the HMU. An extra development cost of having an app in addition to HMU |
| 16 | Attacks | Exposed for MITM attack, an attacker can spoof the identity of the HMU | Exposed for MITM attack, an attacker can spoof the identity of the mobile app and the HMU |

to users it increases the potential attack surface and the risk of the sensitive data being exploited or stolen.

When it comes to the similarities, we have discovered in both options that sensitive fields were shown in plaintext (ID 8). Further, they are both been proven to be exposed for MITM attack (ID 16). It is also important to note that all the items that the HMU is vulnerable for, also is applicable to the hybrid system, as the HMU is not taken out of the equation. This means, i.e. that for the Biotronik system with both app and HMU there are not only the credentials of the mobile app present for an attacker but also the credentials of the HMU. Thus, the attack surface is the largest for the hybrid system, as an attacker has the possibility of exploiting vulnerabilities in both components.

In conclusion, it seems as though having both the HMU and mobile app in a combined system might not be the most beneficial. Both alternatives at the same time may not be a viable long-term solution for Biotronik, as it is the most costly option for the vendor. The added benefits for the patient of being able to see some information from their device do not seem to make up for the added security risks of having two vulnerable devices in the ecosystem. Therefore, we would advise the vendor to move to just one of the two components. This simplifies the ecosystem as a whole and limits the number of devices that Biotronik needs to manage.

8.4 Actual Security vs. Perceived Security From Patients

This section addresses the findings of the questionnaire which can affect the security level of the usage of mobile apps. We put the results from the questionnaire in perspective with the security findings from our analysis. In general, we observed that the patients do not seem very bothered by the security of the applications. The observation implied that they feel as though the benefits of the application weigh up the drawbacks that might exist, as the majority seem very positive about the applications. We discuss some of the findings from the questionnaire that we find interesting.

Increased and Decreased Anxiety

An application linked to a pacemaker has the potential to impact anxiety in patients, both positively and negatively. In the questionnaire, the respondents were asked about their thoughts regarding keeping track of the pacemaker device and its data. With this functionality, patients have their data and can monitor their condition wherever they are at any given time, resulting in reduced anxiety. On the other hand, one respondent from the questionnaire expressed a concern that having an app connected to their pacemaker could increase anxiety for certain individuals. Some

people may feel afraid of what can happen with that information or they can lack the knowledge to understand the data. In addition, the possibility of constantly checking the data since it is available and accessible on their phone might lead some people to develop an obsession.

Patient Reach of the Mobile Application

As discussed in section 7.3, only 7 of the respondents used a mobile application used in the pacemaker ecosystem. Overall, the respondents seem generally positive about the idea of having a mobile app connected to their pacemaker, but still, few patients use the app to this day. Further in the questionnaire, we asked those who do not use the app already, what would make them consider the app. Several respondents answered that a doctor's recommendation made them use the application, which suggests that it is a lack of information regarding the app and its' existence. Additionally, one respondent highlighted that the app was not FDA-approved. This suggests a necessity for better regulation of the apps for patients who value this functionality.

The Value of Health Data

The overall result from the questionnaire suggests that the respondents were not concerned about their security in medical devices. As mentioned earlier, one respondent did not understand why someone would track them, as they perceived themselves as uninteresting. This can show a lack of awareness when it comes to security implications, as health data is extremely valuable for certain groups. According to reports, medical records are worth more than 10 times more than credit card information sold on the dark web [72]. Medical records contain sensitive personal information which can be used for insurance fraud, identity theft, blackmail, etc. Since a pacemaker app collects a lot of data from the patient, it could be very beneficial for an attacker to get a hold of a "normal" person's data, to sell it on the black market.

Password Criteria and Password Habits

From our security analysis of the Biotronik application, we identified the password requirements for the users as weak. In addition, we observed from the questionnaire that the majority of the patients chose to make their passwords by memorable words/phrases. The combination of these two findings results in password generation with a low-security level. If applications have weak requirements for passwords, this can to some extent facilitate the choice of bad passwords. For a service showing health data, which is highly valuable, the passwords should be strong and hard for an attacker to guess.

Misunderstandings

It seems as though the respondents to our questionnaire were concerned about things that do not seem possible to achieve on the applications we have studied in our analysis. For example, some were concerned about the application's ability to accidentally override settings or leave the pacemaker in a "safe mode". For the Medtronic application, which is the only one connecting to the pacemaker directly, it does not seem to be able to communicate with the pacemaker other than to set up a BlueSync connection. However, we are not sure to what extent it would be possible to misuse such a connection to send other commands.

Lack of Security Concerns

The patients generally did not seem to be especially concerned about the security of the pacemaker applications. Several answered that they did not see any drawbacks to using a mobile application connected to their pacemaker. It might be advisable that patients are a bit more mindful of the security of their devices. For example, we have found it possible to make a phishing app that can be authenticated with the Biotronik servers. These types of attacks were not mentioned by any of the users, and it seems as if there is little knowledge of the possibilities.

Request of More Information

Responses to the question of what would make the patients consider using an app with their pacemaker brought forward a wish for more data in the application. It seems as if the general consensus is that the respondents wish for more information about the transmissions being sent, and some more insight into what data is collected from the device. This introduces a trade-off between increased transparency for patients and an increase in security risk by introducing even more information in the apps. The more health data present in the application, the more incentive an attacker has to try and hack these types of applications. This is related to the value of health data presented above. In addition, we can connect even more data to the increased anxiety discussed above.

The patients do not seem to think the security of these applications is very weak. There are some concerns about the theft of data, or being hacked, but generally, they seem positive about the opportunities that the applications bring to their everyday lives. There might be some patients that want stricter restrictions when it comes to the development of these applications. If regulations are introduced, this could be beneficial to increase the number of patients using the application. In addition, the vendors could benefit from running their applications through external security analyses before they are published to the market. This can also give the patients some increased reassurance that the applications keep their data safe.

8.5 Limitations of our Work

As outlined in section 6.1.3, we discovered a restriction within Medtronic. We found a method that implemented a check to determine if the phone was a Samsung phone or not. The first limitation of this research is therefore the lack of a Samsung device to gain access to the Medtronic app. This limitation hinders a comprehensive evaluation of the functionality of the mobile app. We did not have the possibility to obtain a device that was valid for Medtronic's checks throughout the thesis period, which limited our ability to observe the functionality of the device.

If the scope was to be extended, and we were allowed to be able to communicate with the servers, the test cases would have been more flexible and have a bigger test space. Firstly, it would have been possible to evaluate the communication protocols to ensure the confidentiality and integrity of the data transmitted from the mobile app to the server. Further, an analysis of the authentication mechanisms would be conducted, to verify if the implementation was secure. In addition, we could check how the app handles user credentials to the server, to see if the implementation is secure against unauthorized access.

For the questionnaire, there was a potential bias due to the members of the particular Facebook group where the survey was shared. This group consisted of younger patients, and they may have different concerns, expectations, and levels of familiarity with mobile applications and technology compared to older patients. Furthermore, individuals that are a member and participate in such groups are likely to be more concerned about mobile security than individuals that are not part of these groups. Therefore, the group of respondents for our questionnaire is not fully representative of the attitudes of the pacemaker patient group as a whole.

In addition to the aforementioned limitations, we can address our time restrictions. We had to limit the time doing further security analysis, such as analysis with Frida and other dynamic tools. Additional analysis could be useful in the process of accessing the Medtronic app without a Samsung device (section 6.1.3) and the process of bypassing authentication in the Biotronik app (section 6.2.7). If we had the time to explore possible tools a bit further, we might also have found some other tools that could be helpful for further analysis. In addition, we spent a lot of time at the beginning of our work to learn new tools and methods. This time could have been used more effectively if the thesis had been written by someone with more experience doing mobile application analysis.

8.6 Future work

Our work brings valuable insights to the field of mobile applications connected to pacemakers, which has not been analyzed and investigated before. Our contribution is valuable for further research, providing important findings and observations. In addition, we have gained valuable insight into the perspectives of a small group of patients. It is noteworthy that our analysis did not uncover any critical vulnerabilities that pose an immediate and severe threat. However, the research in this thesis has left certain uncovered issues that should be further investigated to ensure the security of the application. These new elements can expose new vulnerabilities, which might cause other attacks. Therefore, we provide this section as future work within our research objectives in the field of mobile applications used in the pacemaker ecosystem.

Repeat the security analysis with a Samsung device: Performing the security analysis on this type of device helps with the compatibility issue for the Medtronic application. If this does not seem possible, one can also set up a Virtual Machine that can root as a Samsung device.

Dynamic analysis with Frida: Continuing on the security analysis with a dynamic analysis tool, such as Frida, provides a deeper understanding of how the app's behavior is during runtime. By dynamically changing the code during the authentication process, it can identify vulnerabilities that possibly exist.

Creating a phishing app: As a result of the client secret being accessible, an interesting and technical experiment could be to design a phishing app to deceive users into downloading it instead of the original, legitimate mobile app. Conducting user tests with patients, under ethical conditions, can reveal potential vulnerabilities in user awareness. This experiment can help to evaluate the effectiveness of countermeasures like training on security and user awareness. It can also determine if additional measures are needed to prevent such phishing attacks.

Further analysis of the mobile applications: As we were not able to successfully bypass the login of the applications, there are a few aspects of their functionality that we have not explored. For example, one can analyze the code base further to improve the understanding of how the application operates, analyze the data storage for a logged-in user, and so on. These aspects should be studied in more detail, to give a broader understanding of the security of these mobile applications.

Perform a more extensive questionnaire: In the future, it could be valuable to obtain a more diverse patient group and perform the same type of questionnaire for them. This gives insight into their habits, which again can indicate whether the patients are improving or worsening the security of the mobile applications connected to their devices.

Chapter 9

Conclusion

The merging of technology and healthcare in recent years has transformed the medical industry, giving several advantages and progress. This includes the integration of applications with medical devices, enabling better patient care, accessible data to the patient, and more efficient medical procedures. In this thesis, we examined two applications that are connected to pacemakers provided by the vendors Medtronic and Biotronik, giving us two different systems. Our research question was the following:

What are the main cyber security concerns for mobile applications in the pacemaker ecosystem when it comes to patient safety and privacy?

Our focus through the thesis was on the potential impact on patients' safety and privacy. To answer this question we did a threat modeling. All the threats are listed in chapter 5. The threat modeling performed revealed potential threats, and several of them were confirmed to be realistic during the security analysis in chapter 8. It included additional security vulnerabilities that were discovered. Overall, the analysis demonstrated how the two different vendors had different approaches when it comes to protecting their applications. The analysis suggests that the main functions of the applications affect which security measures the vendors are implementing. The results of our findings are shown in Table 6.3 for the Medtronic app, and in Table 6.5 for the Biotronik app. The findings reveal attacks where patient privacy could be compromised. The results highlighted the weaknesses in signature algorithms, ineffective password policies, and the risk of downloading a phishing app.

Furthermore, a questionnaire for pacemaker patients was conducted to discover potential security flaws in these two applications. The results of the questionnaire indicated that they generally had a positive view of mobile applications, but their habits and awareness could impact security levels. Furthermore, they had some poor mobile security practices, including weak password habits and a lack of consideration for the potential security consequences of these apps when asked about this.

Our work provides opportunities for further research on these mobile applications on either these vendors or others, and also on other parts of the ecosystem. In addition, further information gathering on patients' perspectives on security can also be conducted.

References

- [1] *<application>*. en. URL: <https://developer.android.com/guide/topics/manifest/application-element> (last visited: Mar. 6, 2023).
- [2] *<permission>*. en. URL: <https://developer.android.com/guide/topics/manifest/permission-element> (last visited: May 18, 2023).
- [3] *Android Permissions - BATTERY_STATS*. en. URL: http://androidpermissions.com/permission/android.permission.BATTERY_STATS (last visited: May 18, 2023).
- [4] *API security best practices | Google Maps Platform*. en. URL: <https://developers.google.com/maps/api-security-best-practices> (last visited: May 13, 2023).
- [5] *Apktool - A tool for reverse engineering 3rd party, closed, binary Android apps*. en. URL: <https://ibotpeaches.github.io/Apktool/> (last visited: Apr. 17, 2023).
- [6] *Arxan Application Protection*. en-US. June 2017. URL: <https://cybersecurity-excellence-awards.com/candidates/arxan-application-protection-2/> (last visited: Apr. 17, 2023).
- [7] Japan Smartphone Security Association. *Android Application Secure Design/Secure Coding Guidebook*. en. 2022. URL: https://www.jssec.org/dl/android_securecoding_en_20220117/6_difficult_problems.html (last visited: May 25, 2023).
- [8] *Authorization - HTTP | MDN*. en-US. Apr. 2023. URL: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Authorization> (last visited: May 9, 2023).
- [9] Ali Balapour, Hamid Reza Nikkhah, and Rajiv Sabherwal. «Mobile application security: Role of perceived privacy as the predictor of security perceptions». en. In: *International Journal of Information Management* 52 (June 2020), p. 102063. URL: <https://www.sciencedirect.com/science/article/pii/S0268401219309041> (last visited: Jan. 31, 2023).
- [10] Elaine Barker and Quynh Dang. *Recommendation for Key Management, Part 3: Application-Specific Key Management Guidance*. en. Tech. rep. NIST Special Publication (SP) 800-57 Part 3 Rev. 1. National Institute of Standards and Technology, Jan. 2015. URL: <https://csrc.nist.gov/publications/detail/sp/800-57-part-3/rev-1/final> (last visited: May 15, 2023).
- [11] *BeVigil - The internet's first and only security search engine for mobile apps*. en. URL: <https://bevigil.com/about> (last visited: May 20, 2023).

- [12] Guillaume Bour, Anniken Wium Lie, et al. *Security Analysis of the Internet of Medical Things (IoMT) - Case Study of the Pacemaker Ecosystem*. en. 2023. URL: <https://guillaumebour.fr/publications/> (last visited: May 24, 2023).
- [13] Guillaume Nicolas Bour. «Security Analysis of the Pacemaker Home Monitoring Unit: A BlackBox Approach». eng. Accepted: 2019-10-18. MA thesis. NTNU, 2019. URL: <https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2623154> (last visited: Oct. 21, 2022).
- [14] Elaine Bzochiewicz, Melissa Chase, et al. «Playbook for Threat Modeling Medical Devices». In: (2021), pp. 20–25.
- [15] *Certificate and Public Key Pinning | OWASP Foundation*. en. URL: https://owasp.org/www-community/controls/Certificate_and_Public_Key_Pinning (last visited: June 4, 2023).
- [16] Deborah Chung. «Materials for electromagnetic interference shielding». en. In: *Materials Chemistry and Physics* 255 (Nov. 2020), p. 123587. URL: <https://www.sciencedirect.com/science/article/pii/S0254058420309500> (last visited: May 6, 2023).
- [17] Catalin Cimpanu. *SHA-1 collision attacks are now actually practical and a looming danger*. en. May 2019. URL: <https://www.zdnet.com/article/sha-1-collision-attacks-are-now-actually-practical-and-a-looming-danger/> (last visited: Mar. 6, 2023).
- [18] *Context - getExternalFilesDir and getExternalCacheDir*. en. URL: [https://developer.android.com/reference/android/content/Context#getExternalFilesDir\(java.lang.String\)](https://developer.android.com/reference/android/content/Context#getExternalFilesDir(java.lang.String)) (last visited: May 15, 2023).
- [19] *Create and monitor geofences*. en. URL: <https://developer.android.com/training/location/geofencing> (last visited: May 15, 2023).
- [20] *CWE - Common Weakness Enumeration*. en. URL: <https://cwe.mitre.org/> (last visited: Feb. 21, 2023).
- [21] *CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')*. en. URL: <https://cwe.mitre.org/data/definitions/89.html> (last visited: Mar. 8, 2023).
- [22] «Cybersecurity Framework». en. In: *NIST* (Feb. 2018). Last Modified: 2022-11-09T11:31-05:00. URL: <https://www.nist.gov/cyberframework/framework> (last visited: Feb. 7, 2023).
- [23] *Dagger basics | Android Developers*. en. URL: <https://developer.android.com/training/dependency-injection/dagger-basics> (last visited: May 13, 2023).
- [24] Tamara Denning, Alan Borning, et al. «Patients, pacemakers, and implantable defibrillators: human values and security for wireless implantable medical devices». en. In: (2010).
- [25] *Difference between Active Attack and Passive Attack*. en-us. Section: Computer Networks. May 2019. URL: <https://www.geeksforgeeks.org/difference-between-active-attack-and-passive-attack/> (last visited: May 4, 2023).
- [26] NHS Digital. *Janus Android Vulnerability*. en. URL: <https://digital.nhs.uk/cyber-alerts/2017/cc-1886> (last visited: Feb. 22, 2023).

- [27] *Environment - getExternalStorageDirectory*. en. URL: [https://developer.android.com/reference/android/os/Environment#getExternalStorageDirectory\(\)](https://developer.android.com/reference/android/os/Environment#getExternalStorageDirectory()) (last visited: May 15, 2023).
- [28] *Field Level Encryption | Couchbase Docs*. en. URL: <https://docs.couchbase.com/sdk-extensions/field-level-encryption.html> (last visited: May 24, 2023).
- [29] *Firebase*. en. URL: <https://firebase.google.com/> (last visited: May 9, 2023).
- [30] OWASP Foundation. *OWASP Top Ten | OWASP Foundation*. en. URL: <https://owasp.org/www-project-top-ten/> (last visited: Feb. 21, 2023).
- [31] *Frida - Welcome*. en-US. Apr. 2023. URL: <https://frida.re/docs/home/> (last visited: May 18, 2023).
- [32] Shivi Garg and Niyati Baliyan. «Comparative analysis of Android and iOS from security viewpoint». en. In: *Computer Science Review* 40 (2021), p. 100372. URL: <https://www.sciencedirect.com/science/article/pii/S1574013721000125>.
- [33] Daniel Halperin, Thomas S. Heydt-Benjamin, et al. «Pacemakers and Implantable Cardiac Defibrillators: Software Radio Attacks and Zero-Power Defenses». en. In: *2008 IEEE Symposium on Security and Privacy (sp 2008)*. ISSN: 1081-6011. Oakland, CA, USA: IEEE, May 2008, pp. 129–142. URL: <http://ieeexplore.ieee.org/document/4531149/>.
- [34] Dick Hardt. *The OAuth 2.0 Authorization Framework*. en. Request for Comments RFC 6749. Num Pages: 76. Internet Engineering Task Force, Oct. 2012. URL: <https://datatracker.ietf.org/doc/rfc6749> (last visited: May 9, 2023).
- [35] *Is MyCareLink Heart safe? | com.medtronic.crhf.mclh BeVigil*. en. URL: <https://bevigil.com/report/com.medtronic.crhf.mclh> (last visited: May 3, 2023).
- [36] *Is Patient App safe? | de.biotronik.PatientApp BeVigil*. en. URL: <https://bevigil.com/report/de.biotronik.PatientApp> (last visited: May 3, 2023).
- [37] *Java Decompiler*. en. URL: <http://java-decompiler.github.io/> (last visited: Apr. 17, 2023).
- [38] Kal. *Exploiting Android's Task Hijacking*. en-us. June 2022. URL: <https://medium.com/mobis3c/android-task-hijacking-6a3a8848f16e> (last visited: May 3, 2023).
- [39] Jakob Stenersen Kok and Bendik Aalmen Markussen. «Fuzzing the Pacemaker Home Monitoring Unit». eng. Accepted: 2021-09-23. MA thesis. NTNU, 2020. URL: <https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2781133> (last visited: Oct. 21, 2022).
- [40] *Learn about using and managing API keys for Firebase | Firebase Documentation*. en. URL: <https://firebase.google.com/docs/projects/api-keys> (last visited: May 13, 2023).
- [41] Statens legemiddelverk. *Glossary – Medical devices - Legemiddelverket*. en. URL: <https://legemiddelverket.no/english/medical-devices/glossary-%5C%E2%5C%80%5C%93-medical-devices#active-implantable-medical-device> (last visited: Feb. 3, 2023).
- [42] Vickie Li. *Hacking JSON Web Tokens (JWTs)*. en. Jan. 2020. URL: <https://medium.com/swlh/hacking-json-web-tokens-jwts-9122efe91e4a> (last visited: Apr. 27, 2023).

- [43] Anniken Wium Lie. «Security Analysis of Wireless Home Monitoring Units in the Pacemaker Ecosystem». eng. Accepted: 2019-10-18. MA thesis. NTNU, 2019. URL: <https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2623147> (last visited: Jan. 31, 2023).
- [44] Wenjun Lin, Ming Xu, et al. «Privacy, security and resilience in mobile healthcare applications». en. In: *Enterprise Information Systems* 0.0 (June 2021). Publisher: Taylor & Francis, pp. 1–15. URL: <https://doi.org/10.1080/17517575.2021.1939896> (last visited: Oct. 14, 2022).
- [45] Muddy Waters Capital LLC. *MW is Short St. Jude Medical (STJ:US)*. en-US. 2016. URL: <https://www.muddywatersresearch.com/research/stj/mw-is-short-stj/>.
- [46] Pere Llorens-Vernet and Jordi Miró. «Standards for Mobile Health-Related Apps: Systematic Review and Development of a Guide». en. In: *JMIR mHealth and uHealth* 8.3 (Mar. 2020). Company: JMIR mHealth and uHealth, Publisher: JMIR Publications Inc., Toronto, Canada, e13057. URL: <https://mhealth.jmir.org/2020/3/e13057> (last visited: Feb. 1, 2023).
- [47] *Location Data | Maps SDK for Android*. en. URL: <https://developers.google.com/maps/documentation/android-sdk/location> (last visited: Feb. 22, 2023).
- [48] Tomomi Matsubara, Masataka Sumiyoshi, et al. «Trend in Age at the Initial Pacemaker Implantation in Patients With Bradyarrhythmia — A 50-Year Analysis (1970–2019) in Japan —». en. In: *Circulation Journal* 86.8 (July 2022), pp. 1292–1297. URL: https://www.jstage.jst.go.jp/article/circj/86/8/86_CJ-21-0947/_article (last visited: May 25, 2023).
- [49] *Medical devices*. en. URL: <https://www.who.int/health-topics/medical-devices> (last visited: Feb. 3, 2023).
- [50] Medtronic. *Cardiac Device Features - BlueSync Technology*. en. URL: <https://global.medtronic.com/xg-en/healthcare-professionals/therapies-procedures/cardiac-rhythm-m/cardiac-device-features/bluesync-technology.html> (last visited: Apr. 6, 2023).
- [51] Medtronic. *MyCareLink Heart App*. en. URL: <https://global.medtronic.com/xg-en/mobileapps/patient-caregiver/cardiac-monitoring/mycarelink-heart-app.html> (last visited: Feb. 22, 2023).
- [52] Medtronic. *Patient Monitoring Solutions | Medtronic*. en. URL: <https://europe.medtronic.com/xd-en/healthcare-professionals/products/cardiac-rhythm/managing-your-patients/remote-management/monitoring-solutions.html> (last visited: May 20, 2023).
- [53] *Medtronic debuts first apps to let heart patients monitor their pacemakers*. en-US. Jan. 2019. URL: <https://venturebeat.com/mobile/medtronic-debuts-first-apps-to-let-heart-patients-monitor-their-pacemakers/> (last visited: Feb. 2, 2023).
- [54] *mitmproxy - an interactive HTTPS proxy*. en. URL: <https://mitmproxy.org/> (last visited: May 4, 2023).
- [55] *Mobile App Experiences (MAX) Explained*. en-US. URL: <https://www.airship.com/resources/explainer/mobile-app-experiences-max-explained/> (last visited: May 16, 2023).

- [56] Jalal Al-Muhtadi, Basit Shahzad, et al. «Cybersecurity and privacy issues for socially integrated mobile healthcare applications operating in a multi-cloud environment». en. In: *Health Informatics Journal* 25.2 (June 2019). Publisher: SAGE Publications Ltd, pp. 315–329. URL: <https://doi.org/10.1177/1460458217706184> (last visited: Oct. 21, 2022).
- [57] NIST. *Glossary / CSRC*. en. Mar. 2023. URL: <https://csrc.nist.gov/glossary> (last visited: June 4, 2023).
- [58] «NIST Retires SHA-1 Cryptographic Algorithm». en. In: *NIST* (Dec. 2022). Last Modified: 2022-12-15. URL: <https://www.nist.gov/news-events/news/2022/12/nist-retires-sha-1-cryptographic-algorithm> (last visited: May 19, 2023).
- [59] *NX bit*. en. URL: https://microsoft.fandom.com/wiki/NX_bit (last visited: May 15, 2023).
- [60] *OAuth vs JWT (JSON Web Tokens): An In-Depth Comparison*. en. URL: <https://supertokens.com/blog/oauth-vs-jwt> (last visited: Apr. 27, 2023).
- [61] *Our History / Digital.ai*. en_US. URL: <https://digital.ai/why-digital-ai/about-us/our-history/> (last visited: May 25, 2023).
- [62] *OWASP MASVS - OWASP Mobile Application Security*. en. URL: <https://mas.owasp.org/MASVS/> (last visited: Feb. 21, 2023).
- [63] *Pacemaker implantation*. en. Section: conditions. Oct. 2017. URL: <https://www.nhs.uk/conditions/pacemaker-implantation/> (last visited: Feb. 3, 2023).
- [64] Bob Pan. *dex2jar*. en. original-date: 2015-03-16T09:13:07Z. Apr. 2023. URL: <https://github.com/pxb1988/dex2jar> (last visited: Apr. 17, 2023).
- [65] *Patient App / Biotronik*. en. URL: <https://www.biotronik.com/en-de/patients/home-monitoring/patientapp> (last visited: Mar. 9, 2023).
- [66] *Reference - android.hardware*. en. URL: <https://developer.android.com/reference/android/hardware/package-summary> (last visited: May 15, 2023).
- [67] *Runtime Permissions*. en. URL: https://source.android.com/docs/core/permissions/runtime_perms (last visited: Feb. 22, 2023).
- [68] Sven Schleier, Carlos Holguera, et al. *OWASP MASTG - OWASP Mobile Application Security Testing Guide*. en. Version 1.5.0. 2022. URL: <https://mas.owasp.org/MASTG/> (last visited: Feb. 2, 2023).
- [69] *SecureRandom*. en. URL: <https://developer.android.com/reference/java/security/SecureRandom> (last visited: May 16, 2023).
- [70] *SHattered*. en. URL: <https://shattered.it/> (last visited: May 19, 2023).
- [71] *SQLCipher README.md*. en. original-date: 2008-07-30T17:20:41Z. Mar. 2023. URL: <https://github.com/sqlcipher/sqlcipher> (last visited: Mar. 8, 2023).
- [72] Brian Stack. *Here's How Much Your Personal Information Is Selling for on the Dark Web*. en-US. Dec. 2017. URL: <https://www.experian.com/blogs/ask-experian/heres-how-much-your-personal-information-is-selling-for-on-the-dark-web/> (last visited: May 30, 2023).

- [73] *Stack Canaries – Gingerly Sidestepping the Cage* / SANS Institute. en. URL: <https://www.sans.org/blog/stack-canaries-gingerly-sidestepping-the-cage/> (last visited: May 19, 2023).
- [74] Android Developer Studio. *Bluetooth permissions*. en. URL: <https://developer.android.com/guide/topics/connectivity/bluetooth/permissions> (last visited: Feb. 28, 2023).
- [75] Android Developer Studio. *Camera API*. en. URL: <https://developer.android.com/training/camera/choose-camera-library> (last visited: Feb. 28, 2023).
- [76] Android Developer Studio. *Data and file storage overview*. en. URL: <https://developer.android.com/training/data-storage> (last visited: Feb. 28, 2023).
- [77] Android Developer Studio. *Shrink, obfuscate, and optimize your app*. en. URL: <https://developer.android.com/studio/build/shrink-code> (last visited: Feb. 27, 2023).
- [78] Yingnan Sun, Frank Po. Lo, and Benny Lo. «Security and Privacy for the Internet of Medical Things Enabled Healthcare Systems: A Survey». en. In: *IEEE Access* 7 (2019). Conference Name: IEEE Access, pp. 183339–183355. URL: <https://ieeexplore.ieee.org/abstract/document/8936335>.
- [79] Tomas Surin. *Inspecting APK Files*. en. URL: <https://pspdfkit.com/blog/2019/inspecting-apk-files/> (last visited: Mar. 3, 2023).
- [80] *Sync Gateway*. en-US. URL: <https://www.couchbase.com/products/sync-gateway/> (last visited: Apr. 28, 2023).
- [81] *The Client ID and Secret*. en-US. Aug. 2016. URL: <https://www.oauth.com/oauth2-servers/client-registration/client-id-secret/> (last visited: May 12, 2023).
- [82] *The GNU C Library (glibc)*. en. URL: <https://www.gnu.org/software/libc/> (last visited: Mar. 10, 2023).
- [83] *Top 10 Mobile Risks - Final List 2016* | OWASP Foundation. en. URL: <https://owasp.org/www-project-mobile-top-10/2016-risks/> (last visited: Feb. 24, 2023).
- [84] Jeanette Tran and Helene Bolkan. *Security Analysis of the Alternatives for the Home Monitoring Unit in the Pacemaker Ecosystem*. en. Project report in TTM4502. Department of Information Security & Communication Technology, NTNU – Norwegian University of Science and Technology, Dec. 2022.
- [85] Sean Turner. *Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms*. en. Request for Comments RFC 6151. Num Pages: 7. Internet Engineering Task Force, Mar. 2011. URL: <https://datatracker.ietf.org/doc/rfc6151> (last visited: May 16, 2023).
- [86] Paweł Weichbroth and Łukasz Lysik. «Mobile Security: Threats and Best Practices». en. In: *Mobile Information Systems* 2020 (Dec. 2020). Publisher: Hindawi, e8828078. URL: <https://www.hindawi.com/journals/misy/2020/8828078/> (last visited: Jan. 31, 2023).
- [87] *What Are Refresh Tokens and How to Use Them Securely*. en. URL: <https://auth0.com/blog/refresh-tokens-what-are-they-and-when-to-use-them/> (last visited: May 3, 2023).

- [88] *What is a public key certificate?* en. URL: <https://www.techtarget.com/searchsecurity/definition/public-key-certificate> (last visited: May 15, 2023).
- [89] *What is Burp Suite?* en-us. Section: Web Technologies. Aug. 2019. URL: <https://www.geeksforgeeks.org/what-is-burp-suite/> (last visited: May 4, 2023).
- [90] *What is MD5 (MD5 Message-Digest Algorithm)?* en. URL: <https://www.techtarget.com/searchsecurity/definition/MD5> (last visited: May 16, 2023).
- [91] *What is the MD5 Algorithm?* en-us. Section: Computer Networks. June 2022. URL: <https://www.geeksforgeeks.org/what-is-the-md5-algorithm/> (last visited: May 3, 2023).
- [92] *What is TLS & How Does it Work? | ISOC Internet Society.* en-US. URL: <https://www.internetsociety.org/deploy360/tls/basics/> (last visited: May 19, 2023).
- [93] Anders Been Wilhelmsen and Eivind Skjelmo Kristiansen. «Security Testing of the Pacemaker Ecosystem». en. MA thesis. NTNU, 2018, p. 167. URL: <https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2609516>.
- [94] Guanglou Zheng, Guanghe Zhang, et al. «From WannaCry to WannaDie: security trade-offs and design for implantable medical devices». English. In: *2017 17th International Symposium on Communications and Information Technologies, ISCIT 2017*. Vol. 2018-January. 17th International Symposium on Communications and Information Technologies, ISCIT 2017 ; Conference date: 25-09-2017 Through 27-09-2017. United States: Institute of Electrical and Electronics Engineers (IEEE), 2017, pp. 1–5.

Appendices

Appendix

Tools and Procedures



A.1 Decompiling an APK File to a JAR File

1. Download apktool and jd-gui with brew with these commands:

```
1 $ brew install apktool
2 $ brew install --cask jd-gui
```

2. Decompile an apk with apktool:

```
1 $ apktool -d <name of the file>.apk
```

3. Download dex2jar on sourceforge ¹
4. Unzip the dex2jar-2.1.zip. Go into the folder dex-tools-2.1 in the terminal
5. Export the classes.dex file into a JAR file with dex2jar:

```
1 $ sh d2j-dex2jar.sh -f ~/path/to/<name of the file>.apk
```

6. Open Java Decompiler in the jd-gui-osx-1.6.6 folder and open the JAR file:

```
1 $ java -jar JD-GUI.app/Contents/Resources/Java/jd-gui-1.6.6-
min.jar
```

A.2 Create and Run an Emulator

An Android emulator is an Android Virtual Device (AVD) representing a specific Android device. The emulator was used to test the mobile applications used in the pacemaker ecosystem.

1. Download Android Studio

¹<https://sourceforge.net/projects/dex2jar/>

2. Go to Tools > SDK Manager and download the preferred Android versions
3. Go to Tools > Device Manager and click on Create device
4. Start the emulator from the command line in the folder `./Library/Android/sdk/emulator/emulator`. See the following command:

```
1 $ ./emulator -avd <name_of_device> -writable-system -no-  
snapshot
```

A.3 How to Install ADB

ADB is a command-line tool used to communicate with an Android device or emulator

1. Go to Android's SDK Platform Tools release notes and download SDK Platform-Tools
2. Go to folder `./Library/Android/sdk/platform-tools` in the terminal
3. Run the following command to check the list of devices attached. When starting the emulator from the command line as in section A.2 above, ADB is automatically connected to the emulator

```
1 ./adb devices
```

A.4 Configure the Emulator to Work with Burp Suite

Burp Suite is used to interact, test and modify the network traffic on the mobile application.

1. Download Burp Suite
2. Go to Proxy and click on Proxy settings
3. Click on Add under Proxy listeners and set Bind to port to 8082 and specific address 127.0.0.1. Click OK
4. On the running Android emulator, click on the three dots for Extended Controls, see Figure A.1
5. Click on Settings > Proxy and choose Manual proxy configuration. Fill in hostname 127.0.0.1 and port number 8082 and click on Apply which will give Proxy status equal Success
6. In Burp Suite, go back to Proxy > Proxy settings. Click on Import/export CA certificate and export Certificate in DER format and save it as cacert.cer

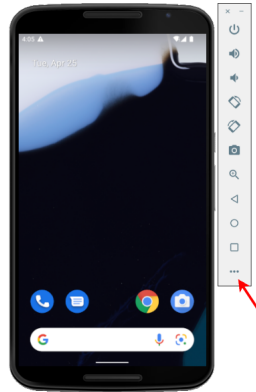


Figure A.1: Emulator showing the option for Extended Controls

7. Install the certificate from Burp Suite on the emulator with these steps in the terminal in the folder `./Library/Android/sdk/platform-tools`:

```

1  $ ./adb root
2  $ ./adb remount
3  $ ./adb push ~/path/to/<name of the file>.cer /system/etc/
  security/cacerts
4  $ ./adb shell chmod 664 /system/etc/security/cacerts/<name of
  pushed certificate>.cer
5  $ ./adb reboot

```

A.5 Configure the Emulator to Work with Mitmproxy

Mitmproxy is a tool for intercepting, modifying, and inspecting network traffic on the mobile application.

1. Download mitmproxy with

```

1  $ brew install mitmproxy

```

2. Run mitmweb (mitmproxy's web-based user interface), bound to port is 8080. Connect the emulator to the proxy as steps 1-5 in section A.4
3. On the emulator, go to mitm.it and download the certificate
4. Execute step 7 in section A.4

A.6 Intercept a Request with Burp Suite

1. Open Burp Suite
2. Go to Proxy > Intercept tab
3. Click on Intercept is off. It will switch to Intercept is on, see Figure A.2

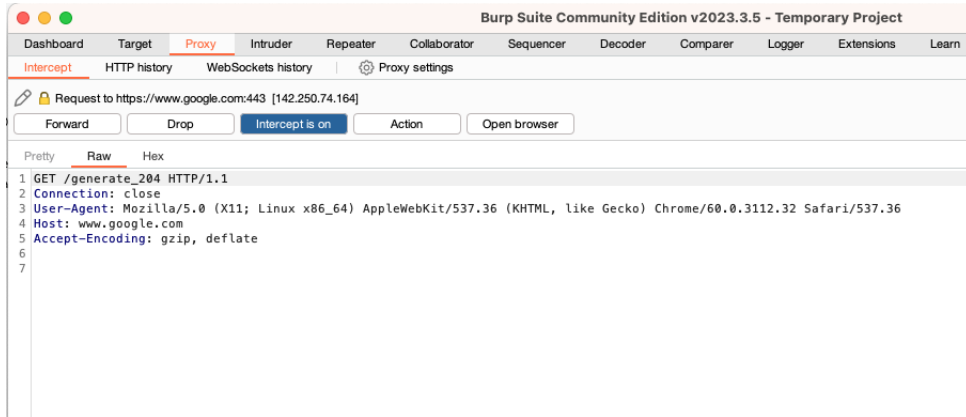


Figure A.2: Toggle the intercept button

4. Go on the emulator and open the mobile application
5. You will see the requests on Proxy > Intercept tab
6. Study and/or modify the request, before forwarding/dropping the request to the server
7. To study and/or modify the response, right-click on the request and forward. Then you will receive the request and modify as you want, see Figure A.3

A.7 Intercept a Request with Mitmproxy

To intercept requests with mitmproxy, a script is necessary to do changes. This involves intercepting a request and returning the desired response instead of sending it to the actual server.

1. Write a mitmproxy script in Python that intercept the requests from the mobile application. See subsection A.7.1 for our script wanting to bypass authentication
2. Run the following command:

```
1 $ mitmweb -s <name on the script>.py
```

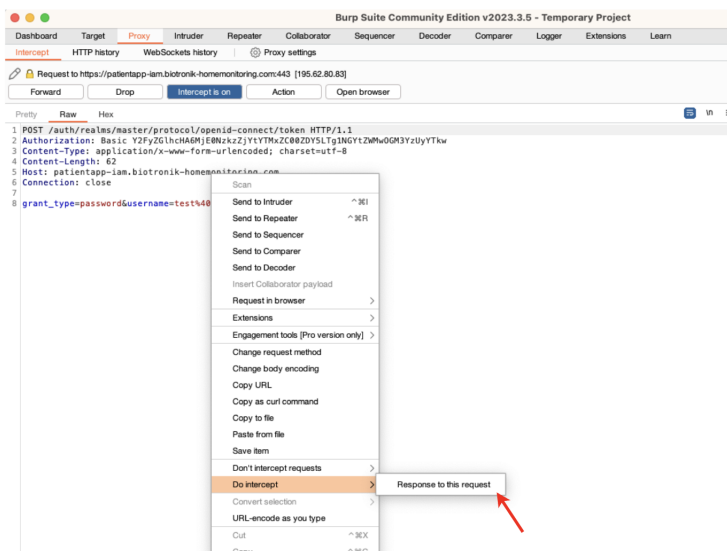


Figure A.3: How to respond to a request

A.7.1 The Script and Corresponding JSON Files

Pythonfile: mock.py

```

1 from mitmproxy import http
2 import os
3 import json
4
5 def request(flow: http.HTTPFlow) -> None:
6     host_auth="patientapp-iam.biotronik-homemonitoring.com"
7     host_session="patientapp-data.biotronik-homemonitoring.com"
8     register="patientapp-server.biotronik-homemonitoring.com"
9     token_path="openid-connect/token"
10    id_path="openid-connect/userinfo"
11    dirpath=os.path.dirname(os.path.abspath(__file__))
12    if host_auth == flow.request.pretty_host and flow.request.path.
13    endswith(token_path):
14        print("request for token")
15        flow.response = http.Response.make(
16            200,
17            open(dirpath + "/accesstoken.json", "r").read(),
18            {"Content-Type": "application/json"}
19        )
20        return
21    elif host_auth == flow.request.pretty_host and flow.request.path.
22    endswith(id_path):
23        print("request for userid")
24        flow.response = http.Response.make(
25            200,

```

```

24         open(dirpath + "/userid.json", "r").read(),
25         {"Content-Type": "application/json"}
26     )
27     return
28 elif host_session == flow.request.pretty_host:
29     print("request for session")
30     flow.response = http.Response.make(
31         200,
32         open(dirpath + "/couchbase.json", "r").read(),
33         {"Content-Type": "application/json"}
34     )
35 elif register == flow.request.pretty_host:
36     print("request for registration")
37     flow.response = http.Response.make(
38         200,
39         open(dirpath + "/register.json", "r").read(),
40         {"Content-Type": "application/json"}
41     )

```

JSON File: accesstoken.json

```

1 {
2     "token_type": "Bearer",
3     "expires_in": 3600,
4     "access_token": "eyJraWQrm8EA4osYg",
5     "refresh_token": "eyJAiYWxnIiA6ICJOb25lIiwgInR5
6     cCIgOiAiSldUIiB9Cg.eyJB1c2VyX25hbWUgOiBhZG1pbiB9Cg"
7 }

```

JSON File: userid.json

```

1 {
2     "user_id": 123,
3     "access_token": "eyJraWQrm8EA4osYg"
4 }

```

JSON File: couchbase.json

```

1 {
2     "couchdb": "Welcome",
3     "vendor": {"name": "Couchbase Sync Gateway", "version": "
4     3.0"},
5     "version": "Couchbase Sync Gateway/3.0.4(13;godeps/)
6     EE"
7 }

```

JSON File: register.json

```

1 {
2   "status": 200
3 }

```

A.8 Setting Up Frida

In order to make Frida work as a tool for dynamic code instrumentation, we need to install it both on our local machines and on the emulator of the phone.

Starting with our local machine, we want to download the main logic of the tool. This was achieved by downloading Frida in our terminal:

```
1 $ pip install frida-tools
```

After successfully downloading the tools onto our machine, we had to install the Frida server for the emulator. First, we downloaded the correct server for our emulator from the official GitHub page.² Thereafter we unpacked this on our machine with:

```
1 $ unxz frida-server.xz
```

The next few steps need to be performed while the emulator is running and with ADB connected to the emulator:

1. Make sure that the emulator is running with root:

```
1 $ adb root
```

2. Export the downloaded server to the emulator:

```
1 $ adb push frida-server /data/local/tmp/
```

3. Make sure that the server has enough access permissions to perform the tasks:

```
1 $ adb shell "chmod 755 /data/local/tmp/frida-server"
```

4. Start running the Frida server in the background on the emulator

```
1 $ adb shell "/data/local/tmp/frida-server &"
```

In order to check that everything is running smoothly, you can run the command

²Frida Github: <https://github.com/frida/frida/releases>

```
1 $ frida-ps -U
```

on your local computer. If everything is configured correctly, this should give a list of processes running on the emulator.

Appendix **B**

Questionnaire

B.1 The Questions

This is the full questionnaire with answer alternatives sent out to the patients.

Demographics

- Age
 - 20-34 35-49 50-64 65-79 80+
- Gender
 - Female Male Nonbinary Prefer not to answer

Clinical Information

- Device Type
 - Pacemaker
 - Implantable Defibrillator (ICD)
 - Cardiac Resynchronization Therapy (CRT)
- Device Manufacturer
 - Medtronic
 - Boston Scientific
 - Abbott / St.Jude
 - Biotronik
 - Other

Mobile Phone Usage

For the following statements, please indicate whether you agree or disagree by selecting the phrase that best describes your view

- I update my phone often
 - Strongly Agree
 - Agree
 - Neither Agree nor Disagree
 - Disagree
 - Strongly Disagree

- I update the applications on my phone regularly
 - Strongly Agree
 - Agree
 - Neither Agree nor Disagree
 - Disagree
 - Strongly Disagree

- I use different passwords for different logins in apps, websites etc.
 - Strongly Agree
 - Agree
 - Neither Agree nor Disagree
 - Disagree
 - Strongly Disagree

- I think about the difficulty of my passwords when I make them
 - Strongly Agree
 - Agree
 - Neither Agree nor Disagree
 - Disagree
 - Strongly Disagree

- I make sure that my presence online is secure by having strong, unique passwords
 - Strongly Agree
 - Agree
 - Neither Agree nor Disagree
 - Disagree
 - Strongly Disagree

- Do you update the applications on your phone automatically?
 - Yes
 - No
 - Not sure

- How do you make your passwords?
 - Random numbers and/or letters
 - Memorable words/phrases

- “suggested” - e.g. from Chrome, Apple etc.
- No consistent method
- Not sure
- Do you store your passwords in a password manager?
- If yes: what type of password manager do you use? I.e. online, offline?

Perceptions of Applications Connected to IMDs

- Do you use an app connected to your pacemaker?
 - Yes
 - No
 - Not sure

For the following statements, please indicate whether you agree or disagree by selecting the phrase that best describes your view

- I think about the security of the medical applications on my phone
 - Strongly Agree
 - Agree
 - Neither Agree nor Disagree
 - Disagree
 - Strongly Disagree
- I think about which permissions an application asks of me, e.g. sharing my location, contacts etc.
 - Strongly Agree
 - Agree
 - Neither Agree nor Disagree
 - Disagree
 - Strongly Disagree
- I am concerned about my data leaking from an application I use
 - Strongly Agree
 - Agree
 - Neither Agree nor Disagree
 - Disagree
 - Strongly Disagree
- I am worried about the possibility of my smartphone being hacked
 - Strongly Agree

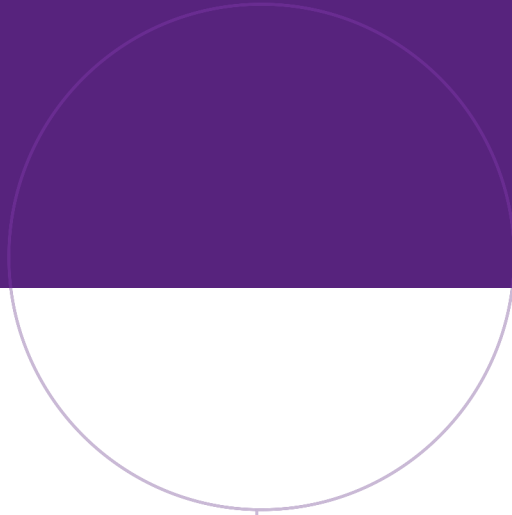
- Agree
 - Neither Agree nor Disagree
 - Disagree
 - Strongly Disagree
- The opportunity to connect my pacemaker to an application on my phone is a good development for my needs
- Strongly Agree
 - Agree
 - Neither Agree nor Disagree
 - Disagree
 - Strongly Disagree
- I use one or more apps to keep track of my everyday health
- Strongly Agree
 - Agree
 - Neither Agree nor Disagree
 - Disagree
 - Strongly Disagree

The following questions are open-ended. You do not have to answer them, but we would appreciate your input on these topics. The answers will help us better understand your perception of security and your thoughts on mobile applications connected to your device.

- What do you think are the benefits of an application connected to your pacemaker?
- What do you think are the disadvantages of an application connected to your pacemaker?
- If you don't use an app already: What would make you consider using such an application?
- What are your thoughts about the opportunity to keep track of your pacemaker device and its data?

Relationship to Cybersecurity

- What are your general thoughts on mobile security?
- Have you ever thought about the security of your medical device?
- Are you aware of any previous security issues related to your medical device?



Norwegian University of
Science and Technology