Security Testing of the Pacemaker Ecosystem

Eivind Skjelmo Kristiansen
Anders Been Wilhelmsen

Master of Science in Communication Technology
Submission date: June 2018
Supervisor: Danilo Gligoroski, IIK
Co-supervisor: Marie Elisabeth Gaup Moe, SINTEF

Norwegian University of Science and Technology
Department of Information Security and Communication Technology

Title:	Security Testing of the Pacemaker Ecosystem
Students:	Anders Been Wilhelmsen and Eivind Skjelmo Kristiansen

Problem Description:

In today's society, we are becoming more and more dependent on our own connected devices. These new devices include not only heartbeat monitors for one's exercise needs, but also vital instruments supporting people's well being. Devices such as insulin pumps and pacemakers have become connected devices which in turn make you, as person, literally connected to the internet. Making everything "smart" could therefore, in the worst-case, have fatal consequences if there is a breach of security for such devices.

In our thesis, the aim is to test the security of the Wireless Body Area Network and medical implants. More specifically we will take a closer look at certain parts of an implantable pacemaker ecosystem and its components, the Home Monitoring Unit (HMU) as well as the programmer. HMUs are small computers that 'talk' with the patient's pacemaker to report back to the hospital with patient data and status. The HMU is not used to control the functioning of the pacemaker. It is only used to transmit data from the pacemaker. If a configuration change of the pacemaker is needed, the patient has to go to a hospital where the pacemaker can be examined and controlled over wireless communication by a larger unit called the programmer.

Because these units are in direct communications with the pacemaker, they pose a potentially severe security risk if compromised. This is why we will test the security of components that belong to the pacemaker ecosystem, in order to look for vulnerabilities that may disclose patient information or pose a threat to patient safety.

Moreover, we aim to do this using only Commercial off-the-shelf equipment for tasks such as disk analysis, network traffic analysis and hardware hacking by for instance connecting to debug ports and reverse engineering of software responsible for communication.

Responsible professor:	Danilo Gligoroski, IIK
Supervisor:	Marie Elisabeth Gaup Moe, SINTEF

Abstract

In later years there has been an increased attention regarding the cybersecurity in ‘smart’ devices. With the rise of the Internet of Things it is becoming popular to connect every ‘smart’ device to the internet, including devices that are responsible for life-critical functions. As such, we, as beings, are also connected to the internet. Over the past few years, there has been an increase in attention especially regarding cybersecurity risks in medical equipment, with companies being sued for performing what is the industry standard practice regarding non-medical devices, namely security testing. With the upcoming of new regulations both in the European Union and the United States, it is important that manufacturers of medical equipment can be trusted to comply with these demands.

In our thesis, we take a deeper look into the state of the security of medical implants with life-critical functions. More specifically, we take a look at the devices inside what is referred to as the *pacemaker ecosystem*, which includes a pacemaker, a pacemaker programmer and optionally a Home Monitoring Unit. Our thesis aims to contribute to the environment of which the pacemaker ecosystem is a part of, meaning both the development cycle of systems that interact with medical devices as well as the environment in which they are utilized. Furthermore, our results will be disclosed to the manufacturer of the equipment that has been tested through the appropriate channels to comply with the principles of Coordinated Vulnerability Disclosure.

Our research is part of a larger project on medical device security at SINTEF led by our supervisor. Our findings provide a thorough evaluation for this particular equipment and resources provided to us, and our results include a platform from which others can continue our work. Furthermore, we use our platform to uncover a large potential attack surface and disclose vulnerabilities which cause concern for patient safety and data privacy. Finally, we suggest countermeasures to the vulnerabilities we discovered, and discuss the potential impact these vulnerabilities could cause. These findings imply that the manufacturer does not comply with the new directives coming from the governing bodies, which regulate these devices in the market and will be mandatory to follow in the near future.

Sammendrag

I senere år har det vært økt oppmerksomhet rundt cybersikkerheten i «smarte» enheter. Med fremveksten av tingenes internett blir det populært å koble alle smarte enheter til internett, inkludert enheter som er ansvarlige for livskritiske funksjoner. Som sådan er vi, som mennesker, også koblet til internett. I løpet av de siste årene har det vært økt oppmerksomhet spesielt rundt cybersikkerhetsrisiko i medisinsk utstyr. Selskaper blir saksøkt for å utføre sikkerhetstesting som er standard praksis for ikke-medisinske enheter. Med kommende nye forskrifter både i EU og USA, er det viktig at at man kan stole på at produsenter av medisinsk utstyr oppfyller disse kravene.

I vår masteroppgave ser vi nærmere på tilstanden til sikkerheten til medisinske implantater som har livskritiske funksjoner. Mer spesifikt undersøker vi enhetene i det som er referert til som pacemakerøkosystemet, som inkluderer en pacemaker, en pacemaker programmerer og eventuelt en hjemmemonitoreringsenhet. Vår oppgave tar sikte på å bidra til miljøet som pacemakerøkosystemet er en del av, noe som både inkluderer utviklingssyklusen til systemer som kommuniserer med medisinske enheter, samt miljøet de benyttes i. Videre vil resultatene bli gitt gjennom passende kanaler til produsenten av utstyret som er testet for å overholde prinsippet om ansvarlig sårbarhetsformidling (Coordinated Vulnerability Disclosure).

Vår forskning er en del av et større prosjekt for medisinsk utstyrssikkerhet ved SINTEF, ledet av vår veileder. Våre funn gir en grundig vurdering av dette utstyret med tilhørende ressurser, og resultatet inkluderer en plattform som andre kan fortsette arbeidet med. Videre bruker vi vår plattform for å avdekke en stor potensiell angrepsflate og sårbarheter som skaper bekymring for pasientsikkerhet og datasikkerhet. Til slutt foreslår vi tiltak mot sårbarhetene vi oppdaget, og diskuterer potensielle konsekvenser av disse sårbarhetene. Disse funnene innebærer at produsenten ikke overholder de nye direktivene som kommer fra styrende organer. Disse vil være obligatoriske å følge i nær fremtid.

Preface

This Master's Thesis is the final deliverable of the Master of Science Degree in Communication Technology with specialization in Information Security at the Department of Information Security and Communication Technology, Norwegian University of Science and Technology (NTNU).

The research is a collaboration between NTNU and SINTEF. It is performed by Anders Been Wilhelmsen and Eivind Skjelmo Kristiansen, supervised by Marie Elisabeth Gaup Moe at SINTEF's research division SINTEF Digital and Danilo Gligoroski at the Department of Information Security and Communication Technology, NTNU.

Acknowledgements

First of all, we would like to thank SINTEF for proposing an interesting and challenging research project with hands-on testing of medical equipment. Even more meaningful and inspiring to us, is the opportunity to contribute to secure the informations system controlling patients' heart rhythm.

The doctors and pacemaker technicians interviewed deserve recognition for their time and willingness to share knowledge of the information system and its environment.

We are also grateful for the support and feedback from SINTEF employees from presentations, over the lunch table, and around the coffee machine. A special thanks to *Terje Frøysa* for helping us with using the SINTEF lab for testing of hardware security.

Éireann Leverett deserves recognition for helping us with security testing and attending a workshop at SINTEF. He has inspired us with his knowledge on security and testing techniques which he has willingly shared. We are very grateful for our numerous discussions and his availability to us.

We would like to thank *Danilo Gligoroski* for being our responsible professor on this project. Danilo has always held is office door open for us, and has provided valuable feedback on the project report.

It is our pleasure to express gratitude to *Marie Elisabeth Gaup Moe*, our supervisor at SINTEF. She has always shown interest for our work, contributed with her knowledge of both devices in the pacemaker ecosystem and its environment. She has helped acquiring equipment, lab access, office space, and established contacts with key resources for the project. She has also been a significant contributor in report quality assurance, regarding language, structure, and content. Marie has been an irreplaceable asset for this project.

Lastly, we are extremely grateful for *Stine Louise Brandal*'s support.

Contents

List of Figures

List of Tables

List of Acronyms

AD	Active Directory.
ASLR	Address Space Layout Randomization.
CIA	Confidentiality, Integrity and Availability.
COTS	Commercial off-the-shelf.
CVSS	Common Vulnerability Scoring System.
DEP	Data Execution Prevention.
EEPROM	Electrically Erasable Programmable Read-Only Memory.
EOS	End of Service.
ER	Emergency Room.
EU	European Union.
FCC	Federal Communications Commission.
FDA	Food and Drug Administration.
GDPR	General Data Protection Regulation.
HMU	Home Monitoring Unit.
HSPA	High Speed Packet Access.
ICD	Implantable cardioverter-defibrillator.
IDA	Interactive Disassembler.
IMD	Implantable Medical Device.

IMDRF	International Medical Device Regulators Forum.
IS	Information System.
IVDR	In Vitro Diagnostic Medical Device Regulation.
JTAG	Joint Test Action Group.
LFSR	Linear Feedback Shift Register.
MDD	Medical Device Directive.
MDR	Medical Device Regulation.
MICS	Medical Implant Communication Service.
NIST	National Institute of Standards and Technology.
NTNU	Norwegian University of Science and Technology.
OS	Operating System.
PCB	Printed Circuit Board.
PGH	Programming Head.
PIC	Position Independent Code.
POTS	Plain Old Telephone Service.
RAM	Random Access Memory.
RF	Radio Frequency.
RSA	Rivest–Shamir–Adleman.
SaMD	Software as a Medical Device.
TLB	Type Library Binary.
UART	Universal Asynchronous Receiver-Transmitter.
UMTS	Universal Mobile Telecommunications System.
USA	United States of America.
USB	Universal Serial Bus.

USRP Universal Software Radio Peripheral.

VM Virtual Machine.

XML Extensible Markup Language.

XOR Exclusive or.

Chapter 1

Introduction

1.1 Background and Motivation

People with certain heart conditions in today's society are living longer and more comfortable due to *Implantable Medical Devices (IMDs)*, like *pacemakers* and *Implantable cardioverter-defibrillators (ICDs)*

A pacemaker is used to monitor the heartbeat and send electrical impulses that cause the heart to contract. It is used to help patients with heart conditions maintain a normal rhythm if they are having trouble with a low or irregular pulse. In the case of an acute cardiac arrest, an ICD can also deliver high energy shocks to kickstart the heart in addition to the pacing functionalities.

Modern pacemakers and ICDs are tiny computers with sensors that detect the heartbeat and physical movement, a computer processor with advanced algorithms that calculate and predict optimal pacing, a memory that stores configuration settings and information about cardiac events, and wireless communication interfaces to allow for remote configuration and monitoring of the device. An ICD is a bit larger than a pacemaker, to accommodate a larger battery and a capacitor, but is still tiny enough to be comfortably implanted under the skin, typically on the chest area, with one, two or three wires called *leads* running inside a vein to the heart. Since the wireless communication interfaces of both pacemakers and ICDs are similar, we will group these two types of devices and refer to them as *pacemakers* for the remainder of this thesis.

As a result of the variety of heart conditions and differences in pacing needs between individual patients, there are multiple configuration options on a pacemaker. These configurations are set by health professionals, typically a cardiologist or nurse with specialized training as *pacemaker technician*, using a device called a *pacemaker programmer* as illustrated in figure 1.1.

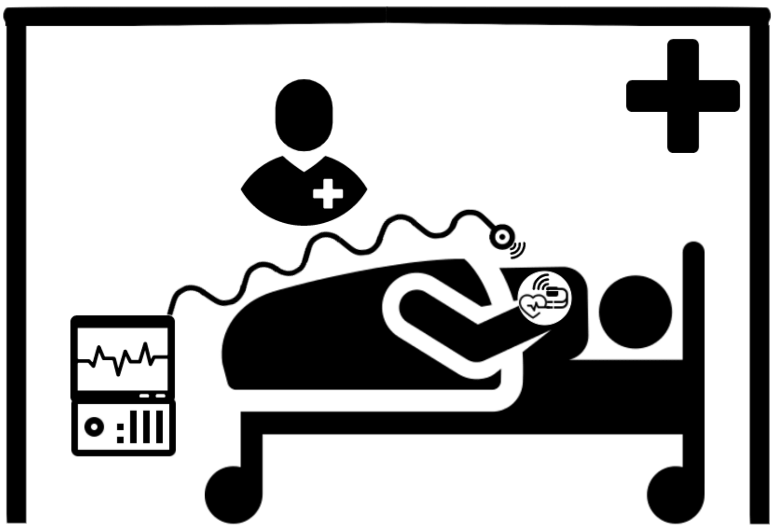


Figure 1.1: The programmer used by a pacemaker technician in the hospital to configure and read data from the pacemaker.

Pacemaker programmers, from here on out referred to as programmers[1], are computers with specialized software that can connect to the pacemaker over a wireless communication interface. Primary uses include interrogating the pacemaker for status information, including statistics of use and details of cardiac events that have been recorded by the device. Also, a pacemaker technician can change or even reset the pacemaker configuration using the programmer.

While the patient is outside the hospital, the pacemaker technician can monitor pacemaker information remotely. This decreases the need for hospital visits, enables the pacemaker technician to detect device issues or abnormal heart rates faster, and provides patients with a sense of security [C$^+$11, CWC$^+$08, STW$^+$07, LPL$^+$12, PLN$^+$12]. Remote monitoring is made possible by using a *Home Monitoring Unit (HMU)* which can interrogate the pacemaker for information and send it to a server hosted by the vendor. This is illustrated in figure 1.2. The information is in turn made available for the pacemaker technician through a website.

[1]Not to be confused with persons who write computer programs.

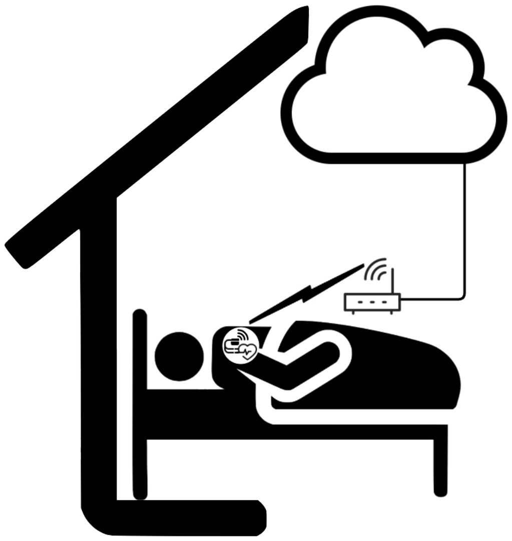


Figure 1.2: The HMU used by patients to monitor their pacemakers and upload data to a cloud service or web server.

Pacemakers, programmers, and HMUs are throughout this thesis referred to as the *pacemaker ecosystem.*

To meet the regulations set by regulatory governing bodies such as the *Food and Drug Administration (FDA)* and *European Union (EU)*, products in the pacemaker ecosystem have a long time-to-market. Ensuring that these devices do not malfunction is a long and rigorous process. However, until recently, these regulations did not take security risks and especially cybersecurity risks into account [Sch16]. Devices in the pacemaker ecosystem created before these regulations included security risks are more likely to include security vulnerabilities. This includes a significant number of pacemakers, HMUs, and programmers that are in use today.

Subsequently our motivation for this topic is anchored in the simple fact that vulnerabilities in the pacemaker ecosystems could be life-threatening.

1.2 Problem Description

Devices in the pacemaker ecosystem are using proprietary software and communicating over proprietary wireless communication protocols. This means that the software controlling a patient's heart and the software used by the pacemaker technician is owned and held secret by the vendor. One consequence of this is a lack of interoperability, causing hospitals to need one programmer from every vendor to be able to connect to all brands and models of pacemakers. Another consequence is that only the vendor can white-box test the security of the software. Furthermore, there are devices in use today that were manufactured before regulations took security risks into account. These devices could have been verified as secure by an independent source, but this is not public knowledge.

1.3 Research Scope

The pacemaker ecosystem has many possible attack surfaces as discussed in our pre-thesis [WSK17]. Including the wireless communication protocol, possible JTAG/UART debug ports in HMUs and the USB-port on the programmer. We chose early not to focus on the pacemaker itself. The decision was made partly because we only had one functional pacemaker at our disposal. Furthermore, knowing that the pacemaker is a proprietary device with an unknown memory structure, we risked spending much time having to learn hardware security and memory forensics. Considering our background, we believed we could produce more tangible results focusing on other parts of the pacemaker ecosystem.

We also chose not to focus on the wireless communication protocol. Also here, we do not have a background in signal processing and therefore concluded it would yield better results if we focused on other parts of the pacemaker ecosystem. Previous studies have also worked on testing the wireless communication protocol, and it seemed to us that we would need training and much time to be able to contribute in this area.

Our initial aim was to identify debug ports on HMUs as described in section 3.2.2. However, the direction changed after failed attempts addressed in section 4.1.2. The focus shifted to an essential device in the pacemaker ecosystem – namely the programmer.

Below is the one functioning programmer at our disposal, with a compatible pacemaker which had battery left.

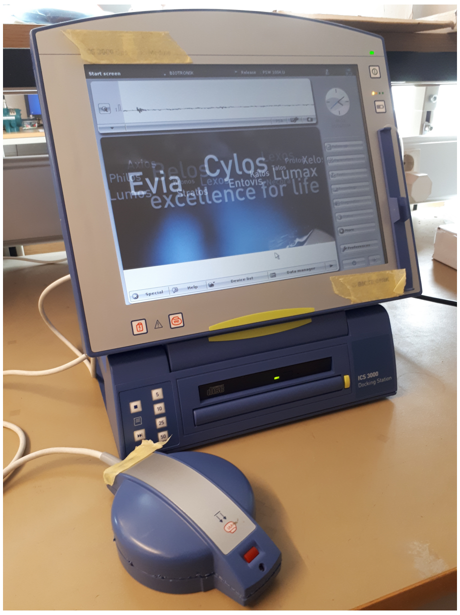


Figure 1.3: The Biotronik ICS3000 programmer at our disposal.

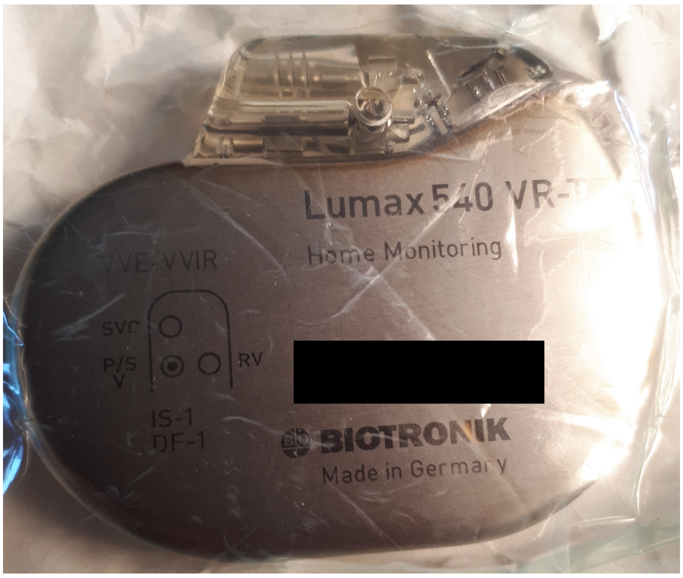


Figure 1.4: The Biotronik pacemaker at our disposal.

As we saw from the pre-study, the programmer is a computer running special software. This is the area of the pacemaker ecosystem that fits our background in information security. Our preliminary knowledge of how the programmer is used has led us to a number of critical questions that are not answered in related literature. There are multiple attack surfaces: on the *Operating System* (OS) level, on the software application layer, and on the programmer interfaces. If the programmer is not connected to the internet, it raises the question of how and when security patching is performed. On the other hand, if it is connected to the internet, what measures have been taken to protect this remotely accessible attack surface? Possible functional requirements may also affect the security of the device. Availability in life-critical situations may affect how or if authentication is implemented. Backward compatibility with legacy pacemakers could leave possible vulnerabilities unfixed. The ability for any programmer to communicate with any pacemaker from the same vendor raises the question how or if encryption and key management are handled. It would also be interesting to unravel how the proprietary software works, which files it contains, what commercial software is used and if executables are signed. The security of the programmer is mostly uncharted territory. As we will discuss in chapter 3, the few available analysis of programmers do not sufficiently verify the security of these devices.

1.4 Hypothesis Statement

The programmer is indeed the key to the heart of a pacemaker patient, due to its central role in accessing the data and configuration of the pacemaker. It is therefore vital that it is secure.

We conjecture that the state of the security in programmers is unknown to the public and that it is possible to study the state of the security in programmers. Specifically, our hypothesis is:

The pacemaker programmer has vulnerabilities that make it insecure in its regular environment.

1.5 Research Question

Based on our motivation, the targeted scope, and problem description we define the following research objectives. The use of *artifacts* and *context* stems from our chosen research method. Artifacts are *'something created by people for some practical purpose'*, for instance software or a model [Wie14]. The context for these artifacts is *'the design, development, maintenance, and use of software and information systems'* [Wie14].

RO.1 Describe *artifacts* in the pacemaker ecosystem

RO.2 Describe the *context* of artifacts in the pacemaker ecosystem

RO.3 Describe vulnerabilities in the pacemaker ecosystem and possible countermeasures.

These objectives are the basis for our research question:

RQ.1 *What is the security of the programmer within the framework of the key principles of Information Security: Confidentiality, Integrity, and Availability (CIA).*

1.6 Summary of Contributions

The main deliverables from this thesis are the knowledge of the state of the security in programmers and the vulnerabilities that have been uncovered. While the mapping of artifacts in the programmer is already knowledge held by the vendor, their implications on the context are unknown to the public.

We have also created a platform from which future work can dive deeper into security testing of programmers as described in section 6.9. The report answers **RQ.1** and addresses the stated research objectives where we describe multiple security issues with the programmer. Security criteria for programmers are suggested, and relevant stakeholders will be informed following principles of Coordinated Vulnerability Disclosure.

1.7 Structure Of Thesis

The remainder of the thesis is structured as follows.

Chapter 2 explains Design Science and justifies why it was chosen as the methodology for this research. It also introduces and argue for the use of qualitative interviews for data collection in the design science framework.

Chapter 3 presents background theory and related literature of security research in the pacemaker ecosystem. This chapter also presents regulatory concerns.

Chapter 4 outlines our results concering the HMU and programmer, and explains how they can be reproduced.

Chapter 5 proposes countermeasures to remedy our findings. It also includes a theoretical proposal for a more secure programmer.

Chapter 6 discusses validation criteria for security and review artifacts and their context in a broader perspective. The chapter evaluates connecting the programmer to the internet, and availability as a requirement. Consequences for people and procedures are discussed, and possible attack vectors and attack scenarios are presented. We argue for the choice of disclosure process before outlining future work.

Chapter 7 concludes the thesis.

1.8 Tools and Resources

In this section, we describe the tools, equipment, and resources used and obtained to improve readability throughout our thesis. Extracting these descriptions simplifies the complexity of the following text, and clarifies what work that has been done. Some prerequisites for the master thesis, which are preliminary results by a pacemaker security project initiated by Dr. Moe, were made available to us.

It is also important to clarify this is not the entire toolkit at our disposal. It is a description of the most necessary tools and resources which might not be known to the reader.

1.8.1 Hardware

JTAGulator

In order to probe and test potential debug ports, one must see if a specific pin acts as expected when guessing what signal it is sending. This process is tedious, and as such, we purchased the JTAGulator[rf]. In short, it automates testing the connected pins, saving time as a reverse engineer.

1.8.2 Software

Hex-Rays IDA Pro

Interactive Disassembler (IDA) Pro is a disassembler and debugger, a program which can pick apart compiled code for reverse engineers to perform code analysis and vulnerability research efficiently. In our project, we use IDA Pro as a tool to understand the libraries of commercial software as well as attempting to understand the binaries created by Biotronik. Since IDA Pro is commercial software, we have only used it for static analysis which is possible with the free version of IDA Pro, available for download at their homepage. As such, our work with IDA should be reproducible without a license.

VirtualBox

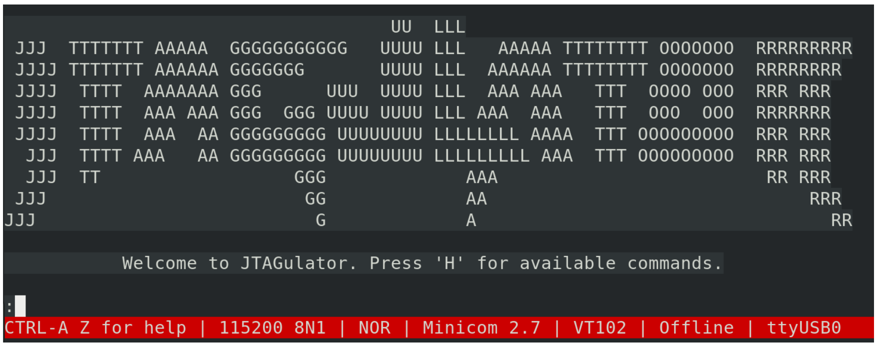
VirtualBox is a freely available program used to create *Virtual Machines (VMs)*. It emulates hardware, making the VM believe it is a normal computer. Using one specific program that is free is important for reproducibility. Although there is a usage of VMware as well, the functionality needed can be performed with the free trial before exporting it in a format that VirtualBox can understand.

dd2vmdk

For converting a disk image to a usable VM disk, an open sourced tool found on GitHub is used. The tool is free and provides us with the necessary conversion to create a VM from our disk image. It was written by the Github user `labgeek`, with his contact email being `vm0x0d@gmail.com`.

minicom

To connect our USB-port to a serial device which is responsible for talking with our JTAGulator, we use the free software minicom which is available as a package for Linux systems.



```
                                    UU  LLL
 JJJ  TTTTTTT AAAAA  GGGGGGGGGGG   UUUU LLL   AAAAA TTTTTTTT 0000000  RRRRRRRRR
 JJJJ TTTTTTT AAAAAA GGGGGGG       UUUU LLL  AAAAAA TTTTTTTT 0000000  RRRRRRRR
 JJJJ  TTTT  AAAAAAA GGG      UUU  UUUU LLL  AAA AAA   TTT  0000 000  RRR RRR
 JJJJ  TTTT  AAA AAA GGG  GGG UUUU UUUU LLL AAA  AAA   TTT  000  000  RRRRRRR
 JJJJ  TTTT  AAA  AA GGGGGGGGG UUUUUUUU LLLLLLLL AAAA  TTT 000000000  RRR RRR
  JJJ  TTTT AAA   AA GGGGGGGGG UUUUUUUU LLLLLLLLL AAA  TTT 000000000  RRR RRR
  JJJ  TT                  GGG             AAA                         RR RRR
 JJJ                        GG             AA                              RRR
JJJ                          G             A                                 RR

           Welcome to JTAGulator. Press 'H' for available commands.

:
CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.7 | VT102 | Offline | ttyUSB0
```

Figure 1.5: JTAGulator welcome screen.

guestmount

A disk image stored in the dd format can be mounted with the `mount` command in Linux. To safely mount a virtual disk image, a `.vmdk` file, a special program called `guestmount` was used on Linux. This has to do with the format of virtual machine disks, which is out of scope for this thesis. It is available as a package for Debian by installing `libguestfs-tools`.

Hivex

Once inside the file system of a virtual machine, we would like to edit specific parameters, such as registry hives. Registry hives are databases used by the Windows system for storing specific settings, and they cannot be edited by Linux trivially since the format is proprietary. Therefore we have to use a special program to edit them on Linux. Hivex was created by reverse engineering the format of how these registry hives are created, to be able to read and write to them. To gain control over the programmer we chose to edit the registry files.

SignTool

SignTool is a tool by Microsoft to sign and verify signatures of files in Windows. SignTool can be executed from the Developer Command Prompt for Visual Studio 2017 using the command `signtool.exe`. Running SignTool in verbose mode with command `signtool.exe verify /v <filename>` displays more details of a signature verification, such as who signed the files as shown below.

```
Verifying: G:\Bio\Execute\Tools\InstImageCD.exe
Hash of file (sha1): 6093FA3E9CFB5682BEF03D7AC194032F0FD95A86
SignTool Error: A certification chain processed, but terminated in a root
     certificate which is not trusted by the trust provider.

Signing Certificate Chain:
    Issued to: BIOTRONIK Issuing CA
    Issued by: BIOTRONIK Root CA
    Expires:   Sat Sep 22 01:36:52 2018
    SHA1 hash: 19BB38EB3977BB9BA9221DFA0DBA2B3D5980835A

        Issued to: Biotronik Programmer
        Issued by: BIOTRONIK Issuing CA
        Expires:   Sat Jun 30 08:53:06 2018
        SHA1 hash: 5CEEA49F231E6FA17B9E51A9B40E267D8789B7DA

File is not timestamped.
```

Listing 1.1: Output of signtool.exe in verbose mode for the executable InstImageCD.exe signed by Biotronik.

1.8.3 Files

Disk Image

Acquired by another member of the pacemaker security project, we have at our disposal a virtual disk image of the contents of the hard drive in our physical programmer.

Patient data

As part of this project, we have access to patient data exported from a programmer belonging to a hospital in the Netherlands. This data has been made available to our research with the consent of the patient, namely our supervisor for this thesis, Dr. Moe. We have access to a pacemaker memory dump in two different versions. Firstly, an encrypted version from a data export from the programmer in the Netherlands, and a redacted version of the same memory dump, but unencrypted. Throughout the thesis, we refer to the encrypted pacemaker memory dump simply as the memory dump, or Random Access Memory (RAM) file.

Biotronik Programmer Software Installers

Software updates for the programmer are available on the Biotronik webpages. The project members have acquired these over time, and accumulate a significant amount of large installer executables. Considering the newer versions from 2015 onwards allegedly has remote update capability, understanding how a software update is performed is also an important part of **RO.2**.

Chapter 2 Method

2.1 Design Science

An *Information System* (IS) is defined as *'a group of components that interact to produce information'*, with the five components being hardware, software, data, procedures, and people [Kro08]. The programmer contains hardware, software, and data. Combined with the procedures of people that are interacting with the programmer in its regular environment, it constitutes an IS. IS research is mainly divided into two paradigms, Behavioral Science and Design Science [CGH09].

Behavioral Science focuses on developing and verifying theories to understand and explain phenomena [BG82, MS95]. This alone does not cover the needs of **RQ.1**, since we do not seek to describe a situation that is observed to exist.

Design Science is another much-used method in IS research [VAMPR04]. The methodology defines *artifacts* and *context*. Artifacts could be methods or software. Hevner describes artifacts as constructs (vocabulary and symbols), models (abstractions and representations), methods (algorithms and practices) and instantiations (implemented and prototype systems) [VAMPR04]. Context is the design, development, maintenance, and use of software and information systems [Wie14].

March and Smith identify *build* and *evaluate* as two iterative processes produced by Design Science research [MS95]. It is illustrated in figure 2.1 as a loop between building new artifacts and evaluating how they affect the environment. Wieringa argues that *'Design Science is the design and investigation of artifacts in context'* and that an artifact should be evaluated in the context they are designed for [Wie14].

As a consequence of being a proprietary system, the artifacts and context of the programmer are to a large extent unknown. This is one of the critical reasons why Design Science relates to the needs of **RQ.1**. As we unravel artifacts from **RO.1**, it is crucial that we can describe how these artifacts affect the context they are in, as stated in **RO.2**. In addition to addressing our need to investigate artifacts in context, Design Science applies to the design of constructs, models, methods, and instantiations to fulfill **RO.1** and **RO.2**. These artifacts can be categorized as instrument design goals since they aid us in answering knowledge questions. Finding vulnerabilities and countermeasures in **RO.3** further relies on the knowledge from investigating artifacts in their context, and is a higher-level design goal that can be categorized as an artifact design goal. Designing artifacts that change the context can also affect the environment, namely people and procedures. However, affecting people and procedures means that we cannot ignore Behavioral Science.

While the goal of Behavioral Science is the truth, the goal of Design Science is utility [VAMPR04]. Lee argues that truth and utility are inseparable [Lee00]. An IS can be divided into a behavioral and technological subsystem that transforms each other in that the phenomena that Behavioral Science describes affects the technological subsystem and vice versa. This separates IS research from technology disciplines and behavioral disciplines, and implies that IS researchers need to master both the technological and the behavioral.

Hevner introduced a framework for IS research, illustrated in figure 2.1, which combine the behavioral science and the design science paradigms.

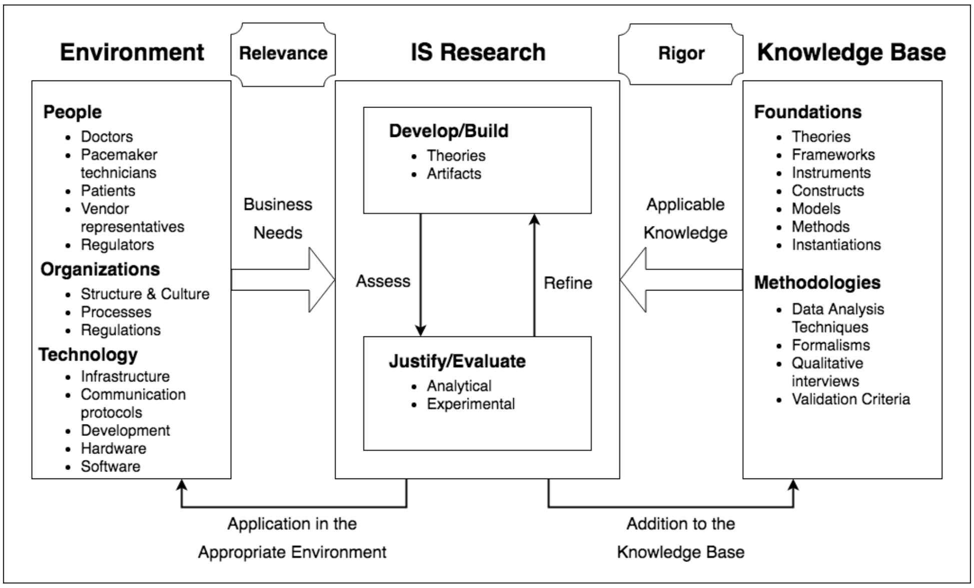


Figure 2.1: Design Science as an Information System Research Framework adapted from Hevner for the pacemaker ecosystem.

The Framework is driven by the motive of improving the environment by introducing new artifacts. The object as stated in **RO.3** is to find vulnerabilities that can affect the environment. These should, in turn, result in countermeasures that will improve the environment. One necessity for **RO.3** are the artifacts designed to fulfill **RO.1** and **RO.2**, which indirectly helps to improve the environment.

Furthermore, it is necessary to justify how our research differentiates from routine design, which is the application of existing knowledge from the knowledge base. Failing to test the state of the security in the programmer uniquely and innovatively, more effective or more efficient, would indicate that we have performed routine design. A clear differentiator between routine design and design research is that is that the research contributes to the knowledge base. Artifacts in context of the programmer is unknown to the public and would be a contribution to the knowledge base. Validation criteria for security in the programmer are process-oriented in recent regulations, and a discussion of these criteria would be another contribution. Knowledge of possible vulnerabilities may also contribute to the design of more secure pacemaker ecosystems, as well as affecting procedures and decisions of patients, pacemaker technicians, and doctors. Furthermore, the utility of different methods and techniques in our research would be a contribution.

Given that Design Science is a problem-solving process, what problem are we trying to solve to answer **RQ.1**? If a problem can be defined as the difference between a goal state and the current state of a system, then we can define the problem as the goal state of the system to be secure, and the current state to be unknown. Revealing the security state of the programmer would be the problem.

Design Science however, addresses what are referred to as *wicked problems* [VAMPR04]. Originally, wicked problems where characterized by ten features identified by Rittel and Webber, with the common factor that wicked problems do not have a clear solution [RW73]. *Tame* problems are in contrast characterized by the clear knowledge of the mission and if the problem is solved. Farrell & Hooker brings nuances to the wicked and tame distinction of problems and reduces the ten wickedness-making features to three conditions [FH13]. Finitude, complexity, and normativity. We use these to argue that **RQ.1** is a wicked problem.

Finitude implies that resources are finite and insufficient for an optimal solution. With a restriction on time, it is not feasible to describe all artifacts in **RO.1**. Thus, **RO.3** is based on a subset of all artifacts. **RQ.1** is therefore based on an incomplete knowledge of the programmer, which should not qualify for an optimal solution. Moreover, there might always exist better solutions to mitigate vulnerabilities.

Complexity implies that processes are difficult to predict, which aggravates the limits imposed by finite resources. The process of discovering the series of actions that constitute a vulnerability is inherently reliant on creativity and trial-and-error since there are a substantial number of possible combinations. Furthermore, viewing the IS as the transformation of a behavioral and technological subsystem as suggested by Lee increases the solution space. We argue that this underpins the complexity of the programmer and that **RQ.1** therefore adheres to the condition of complexity.

Normativity describes how human values and norms require compromise to permit a practicable problem resolution. Normative constraints of **RQ.1** is highly dependent on the values and norms in the programmers regular environment. Although normative constraints may be unknown, it is reasonable that vendors, doctors, and patients have values and norms that require compromises for a practical solution.

We argue that **RO.3** is a wicked problem, in that it adheres to the three conditions stated by Farrell & Hooker. Besides, the solution is dependent on human cognitive abilities such as creativity and teamwork to find vulnerabilities and countermeasures. Furthermore, it is not clear whether the problem is completely solved, whether the security state of the programmer is indeed revealed. There could be undiscovered vulnerabilities or more efficient countermeasures.

It is also necessary to advocate for the relevance of our study. Answering **RQ.1** is not only a question of mitigating possible technical vulnerabilities. It may also affect how vendors work with security in product design, testing during the products lifetime, patching and distribution of patches. Professor Fu at the University of Michigan has been working with manufacturers of devices in the pacemaker ecosystem after demonstrating it was possible to hack an implantable heart defibrillator and deliver a fatal shock in 2008. His claims about improving the security is that *'[..] many manufacturers are still playing to catch up'* and that a small minority were *'ignoring or downplaying the security risks'* [fTDM]. Another indication of the potential for improvement came from Karen Sandler. She is the executive director of the Software Freedom Conservancy and has contributed to excemption for medical device security research in the Digital Millennium Copyright Act (DMCA). A blog post describes her safety concerns regarding her new pacemaker after the publication of multiple pacemaker vulnerabilities coincided with receiving actual threats to her well being.

> As exploit after exploit were published I was sound in the knowledge that at the very least, my device would be safe from remote attack. This became less hypothetical as I (like many other women on the Internet as I have come to understand) have received actual threats to my safety and well being [San17].

In her search for a pacemaker without wireless access, she was met by a Biotronik representative who stated *'Our devices are hack proof'*. A statement which underpins Professor Fu's claim that some manufacturers are downplaying security risks. Furthermore, there is a study of cybersecurity firmware upgrades indicating a potential for improvement in the patching process, which found that pacemaker dependent patients are less likely to receive the security patch and that other factors such as age and sex affected the likelihood of receiving the update [SVE$^+$18].

These indicators for potential of improvement shows that our study is highly relevant.

2.2 Qualitative Interviews

We have chosen to use qualitative interviews as a method for gathering data and hypothesis building in conjunction with the Design Science Framework as depicted in figure 2.1. Qualitative interviews can also be used to uncover normative constraints in the programmer's regular environment. Knowledge of normative constraints are prerequisites for suggesting practical countermeasures that require compromises with values and norms. Viewing the IS as a transformation of a technological and behavioral subsystem, as suggested by Lee, also underpins how qualitative interviews relate to **RQ.1** with regards to the behavioral [Lee00].

The sample has been chosen using the key informant sampling technique, which is set apart from other techniques by sampling candidates with expert knowledge of the domain in question [Mar96]. The selection of candidates is made by the head of a cardiovascular clinic and included in total six doctors and pacemaker technicians working with pacemaker patients on a daily basis. When asked to participate, they were informed of the study's background and purpose, what their participation involves, how we process the information, that participation is voluntary, and that the study has been reported to the Data Protection Official for Research at the Norwegian Centre for Research Data. Data were collected between 11th and 16th of may 2018 with one interview per informant and a contact time of around 60 minutes.

Tremblay describes five criteria for the 'ideal' key informant, which we will use to justify our sample [Tre57]. Since only the first criteria can be answered before the interviews, we will also argue with results from chapter 4.

Role in community implies that the informants' formal role should expose them to the kind of information sought by the researcher. Our informants are exposed to the programmers regular environment which relates to **RQ.1**. They also have insights into how different roles interact with the programmer and which processes are involved, which helps us describe the context of artifacts in **RO.2**. Furthermore, it is reasonable to assume that they have insights of normative constraints that may be unknown to us.

Knowledge means that in addition to having access to the information desired, the informant has absorbed the information meaningfully. As uncovered by the interviews, doctors and pacemaker technicians have gone through extensive theoretical and practical training. This is a clear indication that the informants have absorbed the significance of the desired information.

Willingness means that the informant should be willing to communicate their knowledge to the interviewer and to cooperate as fully as possible. The informants have signed a consent to participate in the study. A concent, however, does not imply willingness. Some interviews had to be rescheduled due to, e.g., implantations and lab work which we interpreted as prioritization of their work rather than lack of willingness. Other informants questioned the necessity of interviews, claiming they had no new knowledge after the interviews of other informants. We interpret this as a sign of data saturation and also prioritization of work, rather than unwillingness.

Communicability suggests that informants should be able to communicate their knowledge in a manner that is intelligible to the interviewer. While medical terms and descriptions of heart anatomy can be ambiguous to us, a more correct understanding was obtained by asking follow-up questions about their meaning. Otherwise, we are reasonably familiar with technical terms regarding devices in the pacemaker ecosystem, and as such the informants were able to communicate in a manner that was understandable for us.

Impartiality describes how key informants should be objective and unbiased, and that any relevant biases should be known to the interviewer. It is reasonable to assume that our key informants are biased to the clinic where they work, and by possible agreements between the clinic and vendors. Doctors are also biased by ethical rules, which in Norway is stated by the Norwegian Medical Association [ftNMA]. Furthermore, we noticed during the interviews that some candidates were aware of others being interviewed, and it is, therefore, reasonable to assume that they might have been biased by each other. Additionally, the fact that we asked questions related to security and data privacy combined with representing a study from two scientific institutions may have affected the informants.

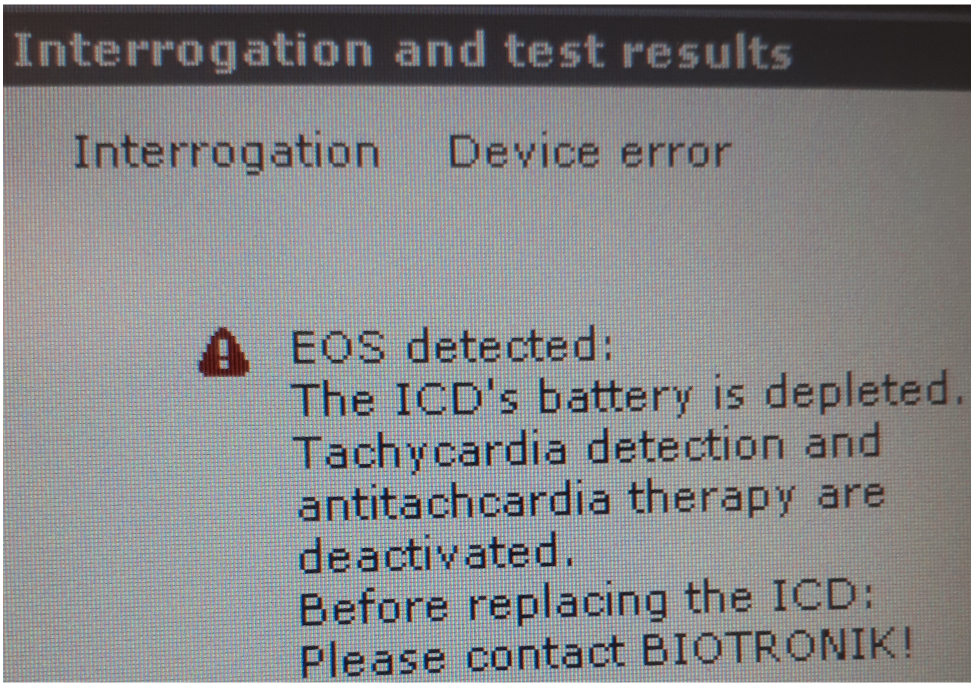
While not being 'ideal' key informants with regards to impartiality, it is apparent that they to a large degree conforms with Tremblay's five criteria.

When justifying a sample size for qualitative interviews, it is necessary to address the concept of *data saturation*. It involves bringing new participants continually into the study until the data set is complete, indicated by data replication or redundancy [MCPF13]. However, we only have a limited number of key informants from one clinic. One possibility would be to reach out to other clinics, but this would require more time and resources. Failing to reach data saturation impacts the quality of the research and hinder content validity [Bow08].

Patton explains that *'Sample size depends on what you want to know, the purpose of the inquiry, what's at stake, what will be useful, what will have credibility, and what can be done with available time and resources'* [Pat90]. We are using key informants because they are believed to have the most knowledge of the programmers regular environment, which is a relatively narrow domain. The information gathered is used for hypothesis building, and as such we are not drawing generalized conclusions based on this research method. The information is further triangulated with documentation and our findings for validation. Charmaz has criticized researchers for using the criterion of saturation to justify very small samples with thin data [Cha11]. We argue that using the key informant sampling technique for hypothesis building in a narrow domain justifies the small sample.

Using the key informant sampling technique, we were able to gather data that relates to the needs of **RQ.1** for insights in the programmers regular environment. The method also meets the needs of **RO.2** since it gives insights into how different roles interact with the programmer, processes involved and allows us to explore further aspects that interview objects find relevant concerning security.

One limitation of our current equipment is that it is not possible to configure our working pacemaker. When the battery level of the pacemaker is almost depleted, the device enters a state called End of Service (EOS), as displayed in Figure 2.2. In this state, the pacemaker can be diagnosed using the programmer, but it is no longer possible to alter the configuration of the device.



Interrogation and test results

Interrogation Device error

EOS detected:
The ICD's battery is depleted.
Tachycardia detection and antitachcardia therapy are deactivated.
Before replacing the ICD:
Please contact BIOTRONIK!

Figure 2.2: EOS warning when interrogating the pacemaker.

This can be remedied by interviewing pacemaker technicians, and doctors, whose knowledge of how a pacemaker can be configured enables us to identify risks to patient safety. Insights into the configuration possibilities and how doctors evaluate the subsequent consequence relates to how a vulnerability can affect patients well being. It is also relevant for discussing what the security level of a programmer should be.

Chapter 3
Related Literature

Before diving into the material, it is important to note that it is little publicly known documentation, analyses, and papers on the field of medical device security regarding the pacemaker ecosystem. As such each piece is described in great detail, providing a thorough understanding of the current state of the art.

3.1 Background Theory

3.1.1 Security Model

We start by describing a security model. The most commonly used is CIA (confidentiality, integrity, and availability). CIA is commonly extended also to include non-repudiation, authenticity, and accountability defined below [Lys15].

Confidentiality ensures that information is disclosed only to those with permission to access it. One example is the encryption of patient data.

Integrity ensures that data has not been manipulated accidentally or by unauthorized people. One example is the signature of files.

Availability ensures that data or services are continuously available to legitimate users.

Non-repudiation ensures that it is possible to prove that an action or transaction was performed by a specific individual so that the action is undeniable. One example is logging of who performed what action on the programmer.

Authenticity ensures the origin of data and correct identification of the sender. One example is that data is signed before being imported by the programmer.

Accountability ensures that an entity can be held accountable since the action can be traced to the responsible. One example is that a doctor cannot deny configuring a pacemaker.

Throughout our thesis, we will address these principles accordingly to the impact of our results, countermeasures, and discussion chapter.

3.1.2 Testing Techniques

There are two main approaches to security testing, namely *black-box* testing and *white-box* testing.

> **Black-box testing** is based on testing the system from a user perspective, without special privileges or insight beyond what is publicly known. This resembles a real attack scenario where an adversary is trying to compromise a system. This approach highlights real security vulnerabilities instead of hypothetical ones and is highly dependent on the skills of the tester. Another disadvantage is that there might exist vulnerabilities that are not discovered.
>
> **White-box testing** is based on using special privileges and knowledge of the source code to look for security vulnerabilities. This is time-consuming and requires detailed insights of the system. However, multiple techniques can be applied, such as static code analysis, automated penetration testing tools and manual penetration testing tools.

Since the pacemaker ecosystem is proprietary, the knowledge of its software and hardware components are kept secret by the vendors. Therefore the black-box testing approach is most applicable to this thesis. However, as stated in section 1.5, our research objectives are not to exploit artifacts but to gain knowledge of artifacts and their contexts to describe vulnerabilities. On this basis, we argue that the application of techniques and methods from the knowledge base in the context of Design Science is the best way of building artifacts as instrument design goals and evaluating artifacts in their context to gain insights.

These methods and techniques include reverse engineering binaries, creating commands and automated scripts to list files and libraries used in executables, and information gathering to understand context. The common factors are that they rely on trial-and-error and creativity to produce results. These factors match our attempts to solve the wicked problem of **RQ.1** since trial-and-error has the features of being problem-specific, solution-oriented and non-optimal. Problem-specific means that there is no attempt to generalize the solution, solution-oriented implies that it only searches for a solution and not why the solution works, and non-optimal means that trial-and-error tries to find one solution, not the best or all solutions.

3.2 Security Testing of Ecosystem Devices

3.2.1 Pacemaker RF Protocol

One of the first papers to touch upon the topic of pacemaker security and privacy directly is Halperin et al. [HHBR+08]. In particular, they take a look at the first generation of pacemakers which has wireless communication capabilities introduced into the U.S. market starting in 2003. Observations done in the paper was performed on the model Medtronic Maximo DR VVE-DDDR #7278 [Data]. From a motivational point of view, the team conducting the testing stemmed from a diverse selection of fields, including personnel actively using the equipment to engineers in both electrical engineering and computer science.

By using the CIA security model for classifying the contributions they can quantify their findings as well as providing important examples of attacks and prototype defense mechanisms. Their choice of method was to rely on COTS equipment that is within the budget of a simple malicious adversary, not necessarily a state actor or organized criminals. Since their findings are based on a single device, their paper act as a catalyst for a larger investigation by the security industry to assess the various devices in the ecosystem.

Regarding passive attacks, using only an oscilloscope they were able to identify and store traces of communication between the programmer and the ICD itself. From there, using known text strings, they were able to identify the modulation and phase of the signal. To obtain these transmissions they used a *Universal Software Radio Peripheral (USRP)*. From here they built an eavesdropper capable of intercepting patient data such as name, date of birth, medical ID number and historical data from several centimeters. As for active attacks, by boosting their signal, they managed to perform several replay attacks from centimeters of distance, being able to change the operating mode of the ICD, change its clock, change the patient name, and inducing fibrillation.

As such Halperin et al. managed to both compromise patient privacy and pacemaker integrity. As for confidentiality, there was never any encryption scheme in place. An important detail to note is that they did not disclose details regarding their constructed attacks which is usually standard practice in security testing, where most do follow Coordinated Vulnerability Disclosure in accordance with ISO/IEC 29147:2014 [ISO14]. Another important point is that they highlight the fact encryption over RF has been implemented before, as a countermeasure to their attacks. One obvious drawback would be the power usage, which is why they have proposed a zero-power key exchange in order to safely introduce encryption over the air at a minimal cost.

In more recent times, another study written by Marin et al. [MSG$^+$16] tests the security of the RF protocol of the same kind of ICD equipment, but for the second generation of pacemakers that have new capabilities such as long-range communication. Longer range communication opens up for more possible attacks and is vital to secure as well. By long range communication, we talk about two to five meters of range, without any specialized equipment that may boost the signal. In a similar fashion to Halperin et al., they also use COTS equipment to perform their attacks, again underlining the fact you do not need to be an entity with a large amount of funds or equipment to compromise these devices.

Again, as is the case with these analyses, there is no access to the source code of how the protocol was set up, so they test using a black box approach. They argue this is more laborsome, yet more realistic due to the fact not everyone having access to open up a device to pick apart its firmware. Following a similar fashion to Halperin et al. they managed to identify the symbol rate, modulation and frequency, although here they simply pointed to the *Federal Communications Commission (FCC)* search page instead of listing the model and make of the ICDs analyzed. They have notified the vendors of the equipment, but have not publicly mentioned in the article which vendor they have been testing the equipment of and reported the vulnerabilities to.

One of the more interesting findings compared to the previous paper is that the symbol seems to have some obfuscation in place, namely an XOR function which picks up its key from a *Linear Feedback Shift Register (LFSR)*. In conclusion, they came to the fact that all ten ICDs analyzed used this encoding scheme coupled with the XOR. This is the first time any obfuscation scheme has been found in the RF protocol between the pacemaker and the programmer. Further on they defined a state model for the energy states to adequately address how to activate the pacemaker for it to be in a state in which it is vulnerable to attacks. As for an active attack, by keeping the device in interrogation mode, they argue this would then drain the battery, or extend the window to perform other malicious activity. To illustrate that these attacks could be done relatively cheap by an adversary, they suggested building a backpack with all necessary equipment to stalk a possible target.

Summarizing their privacy attacks they bear great similarity to the results by Halperin et al., proving that since 2008 no great effort has been made in trying to secure patient data. Again they managed to intercept and read serial numbers, patient name or patient health data if an active session is eavesdropped. In addition to eavesdropping, they also managed to spoof messages and perform replay attacks. Although adversaries might not know the entire format and specification for the protocol, due to there being no replay protection one can send arbitrary commands to the ICD.

In 2016, a report published by Muddy Waters [Blo16] shed light upon the security testing performed by MedSec LLC. It was later independently verified and replicated by Bishop Fox [Liv16]. On attacking through the RF interface they took a different approach compared to the previous papers. Their report concerns devices manufactured and designed by the company formerly known as St.Jude Medical, now Abbott. By compromising the HMU they then managed to, due to re-use of the hardware specifications inside the device between the programmer and their HMU, reprogram it to be able to issue commands that normally would only be done from a programmer to the pacemaker they had in possession.

They then argue that, following this series of exploits, the attack could potentially be upgraded to a large-scale attack, given one compromises either the network that consists of the HMUs or other means, which was redacted from the report. Furthermore, they reason the attack through the HMU could be performed with a range up to ten feet, or three meters, using software defined radios. In addition, they argue that with COTS equipment they could extend that range up to 14 meters, and in special cases for the well-equipped adversary a theoretical limit of 30 meters. Using the access to the HMU they now had their radio already set up, replicating attacks as stated by MedSec LLC. They were able to deliver multiple types of shocks, including false emergency shocks, to the ICD as well as disabling therapeutic functions.

They verified these shocks had really been given by reading the programmer log, reporting shocks above 845 volts. Concerning the communications protocol, and in contrast to the other two papers regarding the radio frequency protocol, they found a seemingly backdoor code for executing commands. Normally, their communication protocol attempted to validate commands from the programmer to the pacemaker using a 3-byte value calculated using RSA, but truncating such a value to 3 bytes provide little to no security with today's computing power. Even then, a unique value always ensured the command was valid. This backdoor value was redacted from the final report. Worse yet, there was no need to replicate the code as they could use the already existing unprotected code from the programmer to perform the 3-byte calculation. Ultimately, combining all these attacks, Bishop Fox agreed that first disabling the therapeutic functions, then performing shocks that could cause cardiac arrest, could lead to a life-threatening situation in which the ICD would not be able to recover the heart functions of a patient.

3.2.2 Home Monitoring Units

Continuing to the security of the devices themselves that the pacemakers communicate with, namely the HMU patients bring with them home to send telemetry data back to the doctor. Previously, we presented the security testing by Bishop Fox provided to prove the claims by MedSec LLC brought forth by Muddy Waters against Abbott. To reuse the hardware as an antenna for the attacks performed on the radio frequency domain, they had to compromise and obtain root access to the device. Following the specific instructions given by MedSec LLC, they were able to connect to certain debug ports called *Joint Test Action Group (JTAG)*, and *Universal Asynchronous Receiver-Transmitter (UART)* granting full access to the device as root, bypassing any operating system access control.

An important distinction is to note that JTAG is a far more primitive yet powerful protocol, and that if enabled it rarely does not provide full access. UART on the other hand can be configured only to provide debug output during, for instance, the boot procedure. This was the case with the Merlin@Home, so the access was only granted on the JTAG interface. From there, on MedSec LLCs instructions they managed to reprogram the device to present a standard Linux root shell on the UART port. Furthermore, they also dumped the unprotected, unencrypted firmware through the JTAG firmware making it available for a complete analysis which could lead to further vulnerabilities being uncovered.

Both attacks which were asked to be reproduced were also done via the Merlin@Home HMU. Not only did Abbott specifically state it was not possible from an HMU in the first place, but that it certainly was not possible to do so without one either. Both statements were proven false, as they replicated both a battery drain attack done under very realistic circumstances: Simulating a human body with flesh, done from a set distance over time, consistently being able to interrogate and drain the battery and a crash attack. The crash attack might be misleading in name, as it does not crash the pacemaker. Instead, it hinders, in some of the attack paths described by MedSec LLC, interrogation between a pacemaker and a programmer. After an unnumbered amount of attempts, the pacemaker will communicate again, but the settings will not be correct as a change in cardiac pacing was observed after the attempted crash attacks. As for the duration it was unresponsive to interrogations, approximately ten days passed before Bishop Fox was able to interrogate with a programmer head again.

Last May 2017, Whitescope released a security evaluation of the ecosystem architecture as a whole [RB17]. By purchasing from public auction sites, Whitescope managed to acquire a total of four vendor subsystems for analysis. Again, it is important to note that Whitescope explicitly leaves out the vendors, such that their results could be disclosed publicly without risking patient health. While not strictly a vulnerability, using COTS chips with readily available datasheets on the internet makes these devices easier for a reverse engineer to work with.

MedSec LLC and Bishop Fox pointed out the same, saying the use of COTS chips could greatly speed up or make it a lot easier for reverse engineers [Liv16, p. 14]. As with the case of Bishop Fox, they found accessible debug interfaces in the form of JTAG and UART on all four vendors, gaining them access to all boxes directly with root privileges. Also discovered by them was the lack of obfuscation, lack of debug symbols, and lack of attempts to slow down a reverse engineer in general for all four vendors.

Moving on to memory access protection, there is no functionality in place to protect the memory range from being intentionally corrupted or overwritten by an adversary, meaning one could alter the functionality of the devices. To make matters worse, they also hardcoded credentials and infrastructure details from the network these HMUs connect to deliver data safely home. While some of these HMUs still connect to the *Plain Old Telephone Service (POTS)*, most of the modern ones today have moved over to 2G or 3G based communication. Still, worldwide many of these devices are still on the old network meaning one can potentially map the authentication network of each vendor where these details are available. In the same manner, but allegedly not with the same explicit detail and instructions, they also reason an attacker with access could continuously interrogate the programmer, effectively launching a battery drain attack. Furthermore, once again without being specific about which vendor, they claim firmware updates are not validated before being installed, resulting in the possibility of a man-in-the-middle attack to intercept and replace the update with a malicious one. This would require quite some effort to do, yet is entirely feasible due to the negligence of the matter. In the same manner, they claim that the software is also not digitally signed, and have no keys stored securely to ensure integrity and authenticity of code running on the HMUs.

3.2.3 Pacemaker Programmers

Continuing on the testing done by Bishop Fox, we shift the focus to the unit most central to our thesis, the *programmer*. Again we would like to underline that this is one of the few official reports with a specific vendor ecosystem, namely Abbott. From their report, it becomes clear few or no security measures were taken in hardening the security of the programmer. Indeed, there has not been implemented any necessary access control on the device such as an encrypted hard drive which would prevent them from modifying the system to obtain a root shell for the device. In the court case, this is described by Muddy Waters as gross negligence of security. Meaning, two very similar approaches were done for obtaining privileged access for the programmer as well as the HMU.

Moving back to the Whitescope paper, as mentioned they got hold of four different vendors programmers for analysis. Again, as was the case with the HMUs, they also found debug ports attached to some of the programmers. Once again they discovered JTAG ports and could obtain access to the entire system. Furthermore, perhaps the most shocking, was their analysis of third-party libraries in which they discovered vulnerabilities from 642 for what they label as vendor four, up to 3715 for vendor two. These are by no means all exploitable or rated as critical, but it shows that there are old libraries in use in critical components responsible for vital functions for patients.

Another similar discovery across all unmentioned vendors was the lack of disk encryption for the removable hard disks inside the programmers, making exfiltration and analysis of the software a lot easier for a reverse engineer. Following up, some of the vendors also had patient data stored unencrypted on the hard disks of the programmers, meaning sensitive data about patient status, logs of their visits, and personal information such as phone numbers and social security numbers were easily accessible to the ones in possession of a disk. Another issue, which might not be regarded as an issue in the eyes of hospital staff or the vendors, is of authentication and availability of the programmer itself and the act of performing programming to a pacemaker. One could argue the issue is to have the machine as available as possible for emergency situations, yet if it lacks basic authentication, as is the case with these unnamed vendors, anyone with any programmer can interrogate and potentially reprogram a pacemaker without any need of a password, secret key or physical smart card. As such, the security of the system is not adequate.

In conclusion, they have discovered many problems that should technically speaking be easy to implement a solution for, yet might compromise availability.

Overall, a clear statement from vendors patching these vulnerabilities has only been done in the case of the court case with Abbott, where two security advisories were released. First, an update that partially fixed the vulnerabilities, then a second patch to follow up released this year in April [ICa, ICb]. Apart from this case, the other reports do not have any advisories or firmware patch notes related to them that indicate cybersecurity vulnerabilities have been fixed. As such it is difficult to conclude whether or not these unspecified vulnerabilities have been addressed at all.

3.3 Regulatory Concerns

Switching topic from the security analyses, Dr. Kevin Fu has written a paper highlighting the lack of security awareness in developing medical devices suited for the technological world of the 21st century, while all focus has been on safety [Fu15]. Where previously medical equipment has been mostly analog, today the trend towards digitalization increases the complexity in a way which breeds insecurity, the paper says. Furthermore, it addresses the state technology enters when compromised, namely unpredictable. As seen clearly in the case of the HMU, Abbott could not, for instance, imagine that the same hardware would be possible to re-use as an attack vector towards the pacemaker itself. Another explicitly mentioned fact here, which has not been previously mentioned, is the example of using an old operating system such as Windows XP. While not covered in this paper because it happened afterwards, we would like to bring attention to the fact that outdated systems are quite common in today's society, as evident by the wannacry incident of May 2017 [New]. During the crisis, computer systems inside hospitals in the United Kingdom were compromised and held ransom, putting lives in danger. As such, hackers and security professionals are regarded as collectors of this so-called technical debt which the paper discusses. Before continuing towards FDAs new regulations it briefly touches upon some of the history of medical devices that have been compromised, not necessarily pacemakers and ICDs. More specifically it mentions Halperin et al., which we have covered in detail previously.

By far, the most important point to bring forth from this paper regarding the state of medical device security was that FDA for the first time include pre-market guidance on cybersecurity as well as the first advisory on avoiding a product as a cause of a cybersecurity risk for an insulin infusion pump. Finally, it also references a guideline to be released by the FDA regarding postmarket security testing of medical devices, and this paper is where our attention will go next.

Probably the most important point to highlight from the postmarket guidelines [Sch16] is the practice of Coordinated Vulnerability Disclosure, meaning to notify vendors of their potential security risks, follow up until it is fixed before publicly disclosing the vulnerability. The FDA has recognized ISO/IEC 29147:2014 Information Technology – Security Techniques – Vulnerability Disclosure as a useful resource for manufacturers.

In the guidance paper, FDA writes:

> FDA recognizes that medical device cybersecurity is a shared responsibility among stakeholders including healthcare facilities, patients, providers and manufacturers of medical devices [Sch16].

While this is true, the largest responsibility is that of manufacturers and hospitals, the ones directly responsible for handling and using the equipment. Since one cannot ensure a device is completely secure for the foreseeable future, it is essential that manufacturers follow certain guidelines for how to tackle issues such as discovered vulnerabilities and how to respond to them. Like in many other regular applications, one has to verify and validate the update process for software for critical systems. One must also use threat modeling to highlight how one shall maintain the security of the device, and at the same time ensure the safety is still intact. Different risks can cause different types of severity in devices, making threat modeling an essential tool for companies risk management strategies. After this, one should then have a plan to:

> protect, respond and recover from the cybersecurity risk [Sch16].

In addition FDA underlines the importance that cybersecurity risks may compromise safety as well, making them inherently critical to fix.

Not only do vulnerabilities need to be addressed, but they also need to be graded in terms of exploitability. FDA suggests a tool which has been through several revisions and is considered one of the industry standards: the Common Vulnerability Scoring System version 3.0, or CVSS v3.0 for short. Through a plethora of numerical ratings, one gets an output of a weighted score which totals the severity of a vulnerability and its exploit potential. As mentioned, there must also be an evaluation of both the severity of patient harm and the risk of patient harm. CVSS v3.0 only assesses the technical parts of a vulnerability, it does not, in this case, apply to a patient relationship with a certain device.

Therefore the FDA has set up some guidelines which have been presented in their paper. For patient severity, they propose a matrix which ranks exploitability versus how controllable the risk might be. As such they separate greatly on uncontrolled and controlled risk to the safety of patients. Uncontrolled risk must follow certain actions as quickly as possible, and are described in the paper. Controlled risk, on the other hand, seems not to be in a state where manufacturers are bound to follow certain steps, but rather encouraged to practice so-called good 'cyber-hygiene'. They define controlled risk for patient safety where the risk is low and acceptable due to the nature of the report.

We would also like to address the newest initiative released by the FDA in April, namely their Digital Health Software Precertification Program [Adm]. In relation to our project, their pilot phase is limited to so-called SaMD, *Software as a Medical Device*, defined by the International Medical Device Regulators Forum (IMDRF) as

> [...] software intended to be used for one or more medical purposes that perform these purposes without being part of a hardware medical device [Gro].

More specifically one of the excellence principles is 'Cybersecurity Responsibility', how committed the vendor is to cybersecurity by proactively addressing them through engagement with stakeholders and peers [Adm, p. 6]. Followingly, device vendors in part of this program can become part of vendors that meet certain standards which would improve their safety and trust from the society. Right now the standard is still in its infancy and the FDA encourages vendors to both answer questions and propose new useful ones to further build a solid foundation for the regulatory body to issue this certificate of excellence. Committed to this program, there is one company with investments in bioelectronics and with the knowledge to create miniature devices which can be implantable. Reported in the media, Verily Life Sciences, formerly known as Google's life sciences, are to develop small electronics which can be implanted that can improve human health [Hir]. Other than this there are no pacemaker ecosystem vendors currently part of the new FDA program.

While not covered in Dr. Kevin Fu's paper, EU has adopted two new regulations on medical devices, MDR 2017/745 and IVDR 2017/746, that will apply in spring 2020 and spring 2022 respectively. They strengthen the post-market surveillance requirements for manufacturers. Article 83 requires all manufacturers to have a post-market surveillance system in place proportionate to the risk class and type of device, and to use the data gathered *'for the identification of options to improve the usability, performance, and safety of the device'*. Furthermore, it requires manufacturers to implement appropriate measures if a need for preventive or corrective action or both are found.

Annex 1 of MDR 2017/745 states that

> All known and foreseeable risks, and any undesirable side-effects shall be minimized and be acceptable when weighed against the evaluated benefits to the patient or user arising from the achieved performance of the device during normal conditions of use [Cou14].

However, medical devices marketed in EU today do not need to meet these regulations. They are regulated by the Medical Device Directive (MDD) 2007/47/EC. Dr. Martin McHugh noted that the last major revision of this standard was made at a time far different from the one we are now in, concerning cybersecurity. Moreover, he continues by urging medical device manufacturers to follow the work of research groups and keep up with industry best practices to protect their devices [Tan].

Norway has a Code of Conduct for information security in the healthcare sector, referenced to as *'the Code'*, which includes suppliers of programmers [fHA]. Following the Code ensures compliance with requirements for information security as provided by Norwegian legislation. The Code specifically targets confidentiality. Stating that only authorized personnel should get access to health and personal information, authorized users should have access in accordance with selected principles, and that access should be logged. For integrity, it states that security measures are to be taken to stop people or technology from changing information without authorization.

3.4 Summary

In this chapter we have looked at related literature, their findings and methods for compromising the security of medical devices, as defined by our security model. Inspired by the approach used by Halperin et al., our paper will also focus more on one specific device and vendor examining the security of the programmer we have in possession from Biotronik. Furthermore, we have presented security measures required to comply with different regulations, which is a prerequisite to discuss if a device is in compliance with regulations.

Chapter 4 Results

In this chapter we present our results of what we have achieved in relation to our research objectives defined in section 1.5. We present our platform for which work by other members of the project can continue from, test the security of the programmer, and discover a vulnerability which compromises patient data confidentiality. We also present artifacts gathered for the system and provide context to artifacts usage and the programmers regular environment.

4.1 Home Monitoring Units

4.1.1 Comparison of HMU Boards

Motherboards or simply boards, are Printed Circuit Board (PCB) connecting electronical components.

When opening our HMUs we can attempt to locate debug ports. By inspecting the boards for areas where there are no connectors soldered on, we can attempt to connect to them with our equipment. Therefore, we present below what we believe could be potential debug ports.

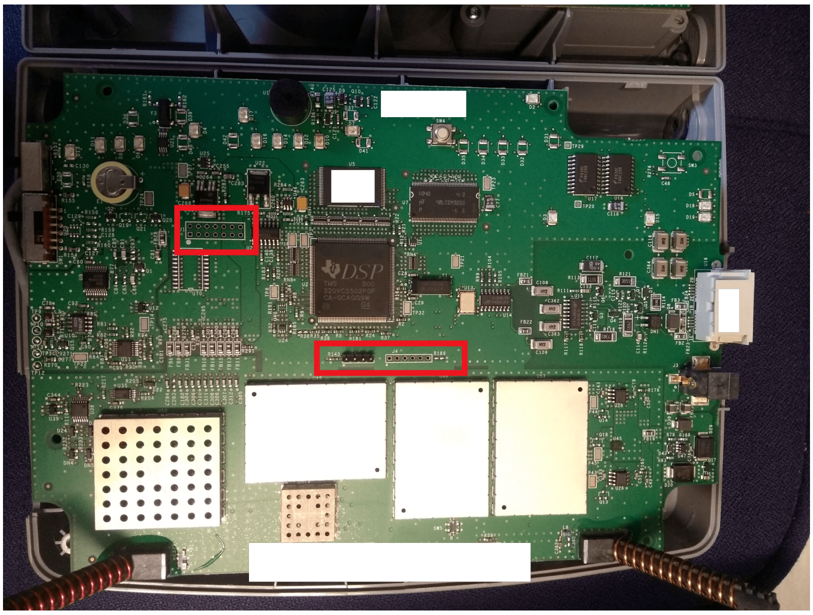


Figure 4.1: First Medtronic board with highlighted potential debug ports, some connectors already soldered on by us.

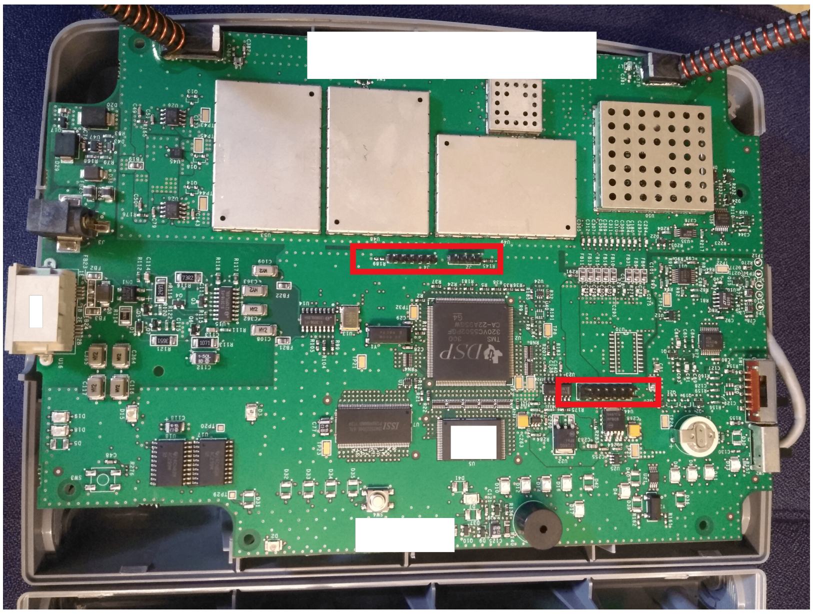


Figure 4.2: Second Medtronic board with highlighted potential debug ports, some connectors already soldered on by us.

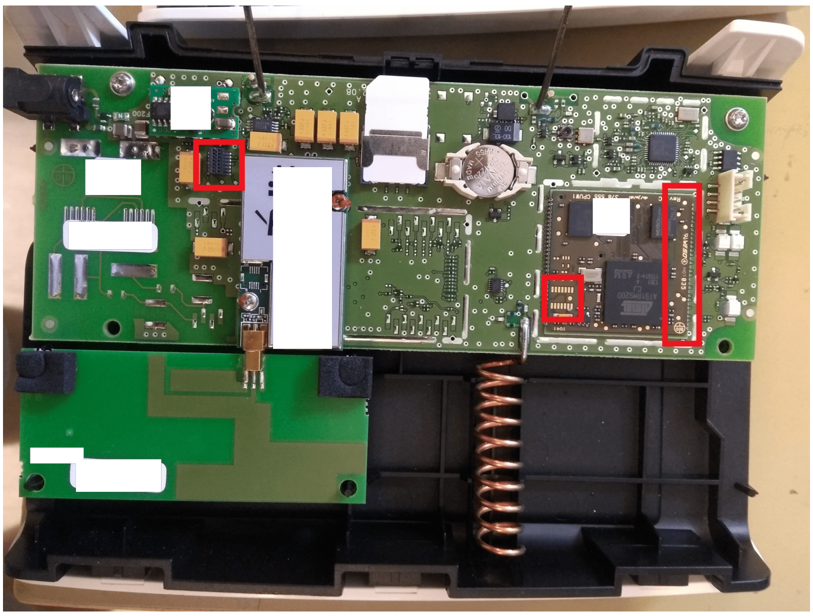


Figure 4.3: First Biotronik board with highlighted potential debug ports.

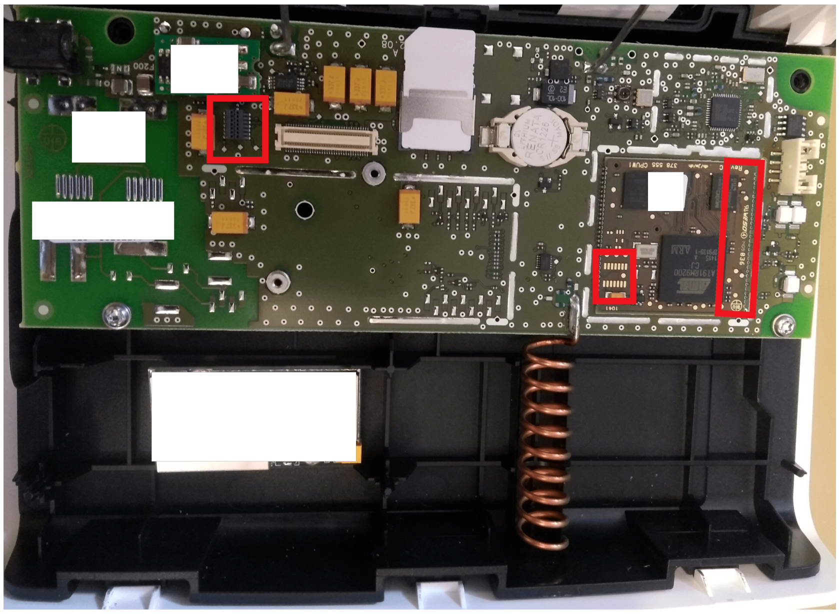


Figure 4.4: Second Biotronik board with highlighted potential debug ports.

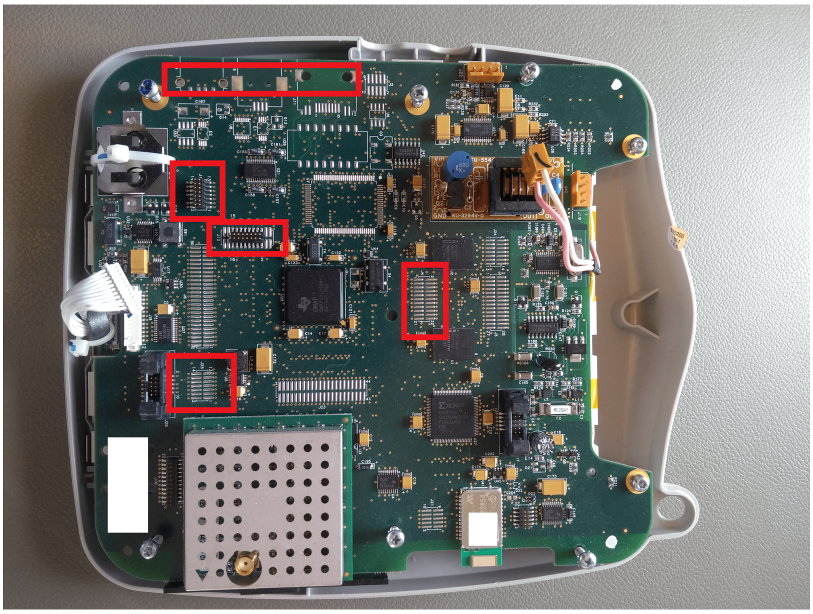


Figure 4.5: Guidant board with highlighted potential debug ports.

Anti hardware reverse engineering efforts As depicted here, we can see there has not been done much effort to hide potential debug ports, giving us clear-cut useful candidates for testing purposes. For Medtronic, it looks as if the ports are merely left open, but unmarked. On the Biotronik ones they look to be more hidden, yet still, have quite a few candidates. Finally, for Guidant, there are multiple possibilities regarding ports, but they seem slightly smaller than the others.

4.1.2 Testing of Debug Ports

In order to work efficiently we used our JTAGulator in the testing process. In the same manner as Whitescope, we attempted to find both JTAG and UART interfaces. Below is the step-by-step process to set up the JTAGulator in order to perform the testing.

1. Find the correct serial device
2. Configure minicom to turn off Hardware Flow Control and set the device to `/dev/ttyUSB0` since we are using a micro USB cable to connect
3. Connect to the board with `minicom -o -c on`
4. Connect the cables to test pins soldered on and start the jtagulating process of identifying pins
5. Follow the steps listed below

Loop to test debug ports using the JTAGulator:

1. Setting the voltage appropriately. If you don't know it, guess.
2. Turning on the board you wish to test
3. Connect to the pins you want to identify
4. Fire up minicom as described
5. Start running the tests using commands from the jtagulator
6. Read output
7. If no ports are found, go to step 3.
8. If you found ports, you are done.

Our first test runs against the Medtronic board gave us connectivity, but the data received was all noise or garbage. Even though we tried to rule out an error on our side by creating new cables, soldering to different pins, and making sure it was not the laptop in use that caused the garbage data, we also swapped to an entirely different Medtronic board. Again, the same kind of garbage appeared. The output data can be found in Appendix F. Resultingly we decided not to chase down this path any further, considering we already spent quite some time without any results coming to fruition. Therefore we moved on to the programmer, assigning this unit our primary focus.

4.2 Pacemaker Programmer

4.2.1 Pacemaker Programmer as a Virtual Machine

Purpose

Before reading these steps, do note that we have created a virtual machine in the OVF (Open Virtualization Format) and exported with all the features below enabled and working. The file is now a part of the project repository.

Creating a virtual machine for the programmer is a vital instrument design goal to answer **RO.1** and **RO.2** for a couple of reasons. Firstly, these programmers are not easy to acquire, and when acquired for a significant amount of money, it makes it even more important not to brick the device when one is working on vulnerability research or testing exploitation methods. Recovering from failure on a locked down physical machine is a lot more difficult than having a virtual environment where one is in full control of the parameters. Furthermore, for a research project involving several people an important objective is to have a common point of view to reliably share information. As such this virtual machine provides the perfect platform to further build upon. Therefore, using only the disk image that was already available to us at the start of this project, we have been able to create a working virtual programmer.

Below follows a list of prerequisites needed and a brief list of steps in order to get a VM to boot with the disk image as a hard drive directly, to allow reproduction of our results.

Reproduction Steps

Prerequisites

1. VMware Workstation 12.x
2. VirtualBox 5.x

3. Virtualbox Oracle VM VirtualBox Extension Pack matching the version number for your VirtualBox installation
4. dd2vdmk tool from github [lab]

Steps for a Baseline Virtual Machine

1. Create a new Virtual Machine inside VMware Workstation 12.x.
2. Select appropriate settings for your virtual machines, living up to the minimum specifications required by Windows XP.
3. Create a vmdk file using the dd2vdmk tool to select as the virtual hard drive for the VM.
4. Start the virtual machine in VMware, and when asked, select yes to upgrade the vmdk format.
5. The VM should now have booted.
6. For a smoother experience, please import the VM into VirtualBox instead.

From this step, you should now have a working VM, although no functional (or barely) mouse, graphics driver, shared folders or networking. From here you need to modify the disk image itself. Before we start modifying the disk image, we decided it is a good idea to clone the VM that has been made to get a VM that is entirely detached from the original disk image. As such, you now have an independent working VM that has no dependency on the disk image, since the disk image, when attached directly to the VM, modifies the file in place. Having a cloned disk rids you of this dependency, and makes it exportable as well. From there, a couple of things needs to be installed and done. For us to get anything installed such as VirtualBox guest tools we need to obtain control over the system. Since the hard drive is not encrypted, we can modify its contents to do what we want. Therefore, with as little intrusion as possible, following the below steps grants you access to a command prompt inside the virtual machine from which you will perform a number of steps to get a much more comfortable environment to work in. The next list of prerequisites is a supplement to the previous one.

Prerequisites

1. hivex [Jona]
2. guestmount [Jonb]
3. A text editor of your choice

Steps for an Enhanced Virtual Machine

1. Clone your virtual machine in side VirtualBox to create a separate vmdk that has no relation to the original disk image.

2. With the VM off, mount the D: partition of the vmdk using `guestmount`:
   `sudo guestmount -a <path to .vmdk> -m /dev/sda2:/:remove_hiberfile, \`
   `rw,ntfs-3g --rw <mountpoint>`

3. As root, navigate to `WINDOWS\system32\config` and open the registry hive `SOFTWARE` with the following command: `hivexsh -w SOFTWARE`
   Navigate to the following folder: `SOFTWARE\Microsoft\Windows\CurrentVersion\Run`.
   You should now see the following output if you run `lsval`:

```
SOFTWARE\Microsoft\Windows\CurrentVersion\Run> lsval
"Autostart"="d:\\bio\\execute\\instimaged.exe"
"tsharc"="D:\\Program Files\\tsharc\\hwincal.exe -usecustom"
"CHIPSStart"="CHPSTART.EXE"
"IgfxTray"="D:\\WINDOWS\\system32\\igfxtray.exe"
"HotKeysCmds"="D:\\WINDOWS\\system32\\hkcmd.exe"
"BluetoothAuthenticationAgent"="rundll32.exe bthprops.cpl,,
   BluetoothAuthenticationAgent"
```

4. Perform the following steps to overwrite the programs to be run at boot with a command prompt:

```
SOFTWARE\Microsoft\Windows\CurrentVersion\Run> setval 1
key> Shell
value> string:C:\Windows\system32\cmd.exe
SOFTWARE\Microsoft\Windows\CurrentVersion\Run> lsval
"Shell"="C:\\Windows\\system32\\cmd.exe"
SOFTWARE\Microsoft\Windows\CurrentVersion\Run> commit
SOFTWARE\Microsoft\Windows\CurrentVersion\Run> q
```

5. Perform `sudo guestunmount <mountpoint>` and boot your virtual machine. You should be able to alt-tab to a command prompt.

6. Once in the command prompt, type `explorer.exe` to open the file browser. From here, you install the guest addition tools for VirtualBox and restart.

7. Upon reboot, select the proper screen resolution through `explorer.exe` and by typing `Control Panel` in the navigator bar.

8. Open up device manager through cmd by typing `devmgmt.msc` and install the driver for the unknown USB device with a question mark on it. The driver you are looking for will be a
   `Standard OpenHCD USB Host Controller`. Reboot. USB drives should now work when passed to the virtual machine.

9. To get shared folders working, share a folder in VirtualBox the usual way. It will not appear automatically, but by typing `\\vboxsvr` in the navigator bar you should see it listed.

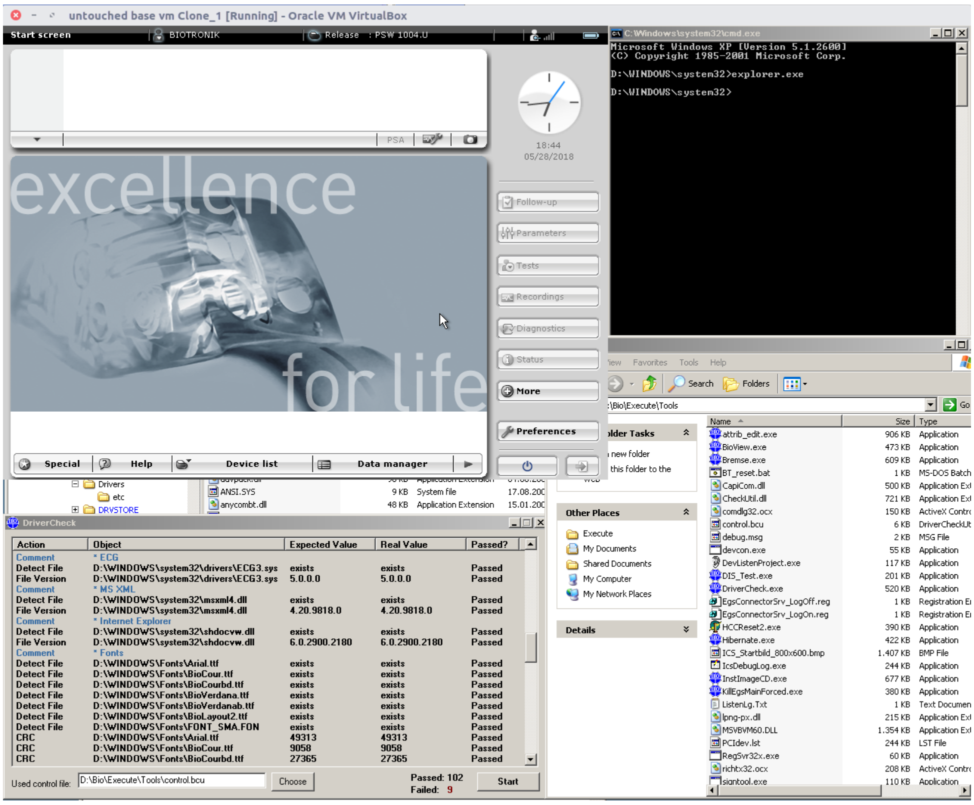
Finally to get basic network functionality to work with a static IP, perform the following steps:

1. Download the driver pack `PRO2K3XP_32.exe` from [Int] and extract it to a folder.
2. Copy this folder over to the VM through the shared folder system previously set up.
3. Change the adapter to `Intel (R) PRO/1000MT Network Connection` in VirtualBox, reboot the VM.
4. Manually point the driver installation for the network adapter in `devmgmt.exe` to the folder which contains the correct driver in the folders you uploaded in step 1.
5. Reboot when prompted, then open the Control Panel on reboot and open Network Connections. It might take some time before the adapter appears properly.
6. Right-click the connection, select `properties` and configure a static IP address inside your host-only configured virtual network from VirtualBox.

These steps are the ones that should be followed in order to reproduce the VM we have created and exported for further use in this research project.

VM as Platform for Further Work

Getting a fully functional virtualized programmer is a key contribution of our work that allows other researchers to build upon our results in the larger project. Full control of a virtual copy of the programmer opens new paths that can be explored. Figure 4.6 is a screenshot of the VM. On the top-left is the only software available to doctors and pacemaker technicians, but there are now new possibilities with internet connectivity, shared folders with the host machine, and an interactive shell providing access to system utilities, the file system and software of our choice.



```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

D:\WINDOWS\system32>explorer.exe

D:\WINDOWS\system32>
```

ANSI.SYS	9 KB	System file	17.08.200	
anycombt.dll	48 KB	Application Extension	15.01.200	

Action	Object	Expected Value	Real Value	Passed?
Comment	* ECG			
Detect File	D:\WINDOWS\system32\drivers\ECG3.sys	exists	exists	Passed
File Version	D:\WINDOWS\system32\drivers\ECG3.sys	5.0.0.0	5.0.0.0	Passed
Comment	* MS XML			
Detect File	D:\WINDOWS\system32\msxml4.dll	exists	exists	Passed
File Version	D:\WINDOWS\system32\msxml4.dll	4.20.9818.0	4.20.9818.0	Passed
Comment	* Internet Explorer			
Detect File	D:\WINDOWS\system32\shdocvw.dll	exists	exists	Passed
File Version	D:\WINDOWS\system32\shdocvw.dll	6.0.2900.2180	6.0.2900.2180	Passed
Comment	* Fonts			
Detect File	D:\WINDOWS\Fonts\Arial.ttf	exists	exists	Passed
Detect File	D:\WINDOWS\Fonts\BioCour.ttf	exists	exists	Passed
Detect File	D:\WINDOWS\Fonts\BioCourbd.ttf	exists	exists	Passed
Detect File	D:\WINDOWS\Fonts\BioVerdana.ttf	exists	exists	Passed
Detect File	D:\WINDOWS\Fonts\BioVerdanab.ttf	exists	exists	Passed
Detect File	D:\WINDOWS\Fonts\BioLayout2.ttf	exists	exists	Passed
Detect File	D:\WINDOWS\Fonts\FONT_SMA.FON	exists	exists	Passed
CRC	D:\WINDOWS\Fonts\Arial.ttf	49313	49313	Passed
CRC	D:\WINDOWS\Fonts\BioCour.ttf	9058	9058	Passed
CRC	D:\WINDOWS\Fonts\BioCourbd.ttf	27365	27365	Passed

Used control file: D:\Bio\Execute\Tools\control.bcu

Passed: 102
Failed: 9

:\Bio\Execute\Tools

Name	Size	Type
attrib_edit.exe	906 KB	Application
BioView.exe	473 KB	Application
Bremse.exe	609 KB	Application
BT_reset.bat	1 KB	MS-DOS Batch
CapiCom.dll	500 KB	Application Ex
CheckUtil.dll	721 KB	Application Ex
comdlg32.ocx	150 KB	ActiveX Contr
control.bcu	6 KB	DriverCheckUt
debug.msg	2 KB	MSG File
devcon.exe	55 KB	Application
DevListenProject.exe	117 KB	Application
DIS_Test.exe	201 KB	Application
DriverCheck.exe	520 KB	Application
EgsConnectorSrv_LogOff.reg	1 KB	Registration E
EgsConnectorSrv_LogOn.reg	1 KB	Registration E
HCCReset2.exe	390 KB	Application
Hibernate.exe	422 KB	Application
ICS_Startbild_800x600.bmp	1.407 KB	BMP File
IcsDebugLog.exe	244 KB	Application
InstImageCD.exe	677 KB	Application
KillEgsMainForced.exe	380 KB	Application
ListenLg.Txt	1 KB	Text Documen
lpng-px.dll	215 KB	Application Ex
MSVBVM60.DLL	1.354 KB	Application Ex
PCIdev.lst	244 KB	LST File
RegSvr32x.exe	60 KB	Application
richtx32.ocx	208 KB	ActiveX Contr
signtool.exe	110 KB	Application

Figure 4.6: Screenshot of VM with the only software available on the physical programmer, a shell, file browser and a Biotronik testing tool.

4.2.2 List of Files

Listing files on the disk is an instrument design goal to answer **RO.1**. Creating an overview of the file structure also gives insights to **RO.2**. This was addressed with the scripts in Appendix A that parses a disk image for files, verifies file signatures, calculates their hash value and stores the information in a LaTeX table with the file path and comments.

Reproduction Steps

Prerequisites

1. VirtualBox 5.x
2. Windows 7

3. Visual Studio 2017
4. Disk image `ICS3k.001`

Setup for Script

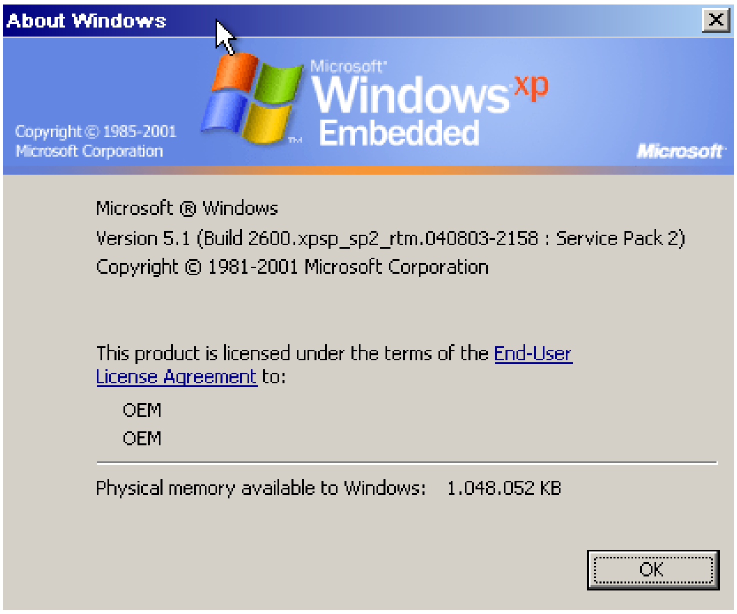
1. Create a new Virtual Machine in VirtualBox 5.x with Windows 7
2. Install Visual Studio 2017
3. Find location of `signtool.exe` and add the folder to path variable with `set PATH="C:\Program Files\Microsoft SDKs\Windows\v7.1\Bin";%PATH%`
4. Install Python 2.7 and add it to path variable
5. Install pip and add it to path variable
6. Install dependencies for script with `pip install -r requirements.txt`
7. Download and install Microsofts CAPICOM SDK v. 2.1.0.2. Copy capicom.dll from `\Program Files\Microsoft CAPICOM 2.1.0.2 SDK\Lib\X86` to `\Windows\System32` and execute `REGSVR32 capicom.dll` from a command prompt with administrator rights.

Mount Disk Image in VirtualBox

1. Convert `ICS3k.001` to VDI file with command `VBoxManage convertdd ICS3k.001 ICS3kVDI.vdi --format VDI --variant Fixed`
2. Mount disk as a drive in VirtualBox
3. Change permissions on all files on mounted drive with command `icacls "G:" /grant IEUser:G /T` in Windows 7. Granting user IEUser read, write and execute permissions recursively on mounted G: drive.

Interesting Files

To filter out files that are likely not to be of interest from the total of 14067 files on both partitions, the script excludes files that match the hash value or file path and name with that of a standard Windows XP installation. To achieve this, we have a custom script for Windows XP edited for readability in Appendix A.1 listing A.1. The programmer uses Windows XP with Service Pack 2 as shown in figure 4.7. Therefore, the mapping of files is performed after a clean installation of Windows XP with Service Pack 2.



Microsoft ® Windows
Version 5.1 (Build 2600.xpsp_sp2_rtm.040803-2158 : Service Pack 2)
Copyright © 1981-2001 Microsoft Corporation

This product is licensed under the terms of the End-User License Agreement to:

OEM
OEM

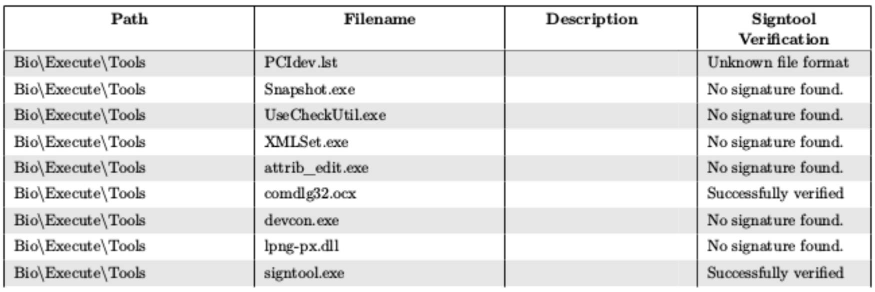
Physical memory available to Windows: 1.048.052 KB

Figure 4.7: 'About Windows' showing that the programmer is running Windows XP with Service Pack 2.

Filtering out default file paths and file names exclude only 325 files (2,6 %) on the D: partition and 421 files (28,3 %) on the C: partition. Filtering on file hashes, however, filter out 6952 files (55,3 %) on the D: partition and 171 (11,5 %) files on the C: partition. In total this approach filters out 55,9 % of files which are reasonable to believe as default system files for Windows XP.

SignTool

SignTool addresses **RO.1** by verifying which files are signed and not. It is used by the script, and results of verification for each interesting file are added to the table as shown in figure 4.8.



Path	Filename	Description	Signtool Verification
Bio\Execute\Tools	PCIdev.lst		Unknown file format
Bio\Execute\Tools	Snapshot.exe		No signature found.
Bio\Execute\Tools	UseCheckUtil.exe		No signature found.
Bio\Execute\Tools	XMLSet.exe		No signature found.
Bio\Execute\Tools	attrib_edit.exe		No signature found.
Bio\Execute\Tools	comdlg32.ocx		Successfully verified
Bio\Execute\Tools	devcon.exe		No signature found.
Bio\Execute\Tools	lpng-px.dll		No signature found.
Bio\Execute\Tools	signtool.exe		Successfully verified

Figure 4.8: Part of table created by the script to list interesting files

It appears from the list of files that many files are unsigned. Consequently, there are no procedures to verify the integrity of these unsigned files. Launching `EGSStart.exe` instantiates `EgsMain.exe` which is the only software visible for doctors and pacemaker technicians. This is central file that is unsigned, thereby lacking integrity protection.

Log Files of Programmer Usage

Finally, there are also stored log files of every action and button press done for each session on the programmer in XML format in `D:\Bio\Execute\Log`. We did not find a way to view these in the programmer software, but accessed them through the filebrowser in the VM.

4.2.3 Commercial Software

Description of commercial software and proprietary software stored and used by the programmer addresses **RO.1**. Understanding what function they have and what they do also addresses **RO.2**. Further insights of their capabilities and known vulnerabilities are relevant for **RO.3**.

Kithara

Kithara is a proprietary Real-Time System for Windows which *'creates its own protected area, that is independent and unrestricted by the operating system'* [Soff]. The programmer is running version 6.03d which is old considering that the oldest entry in Kitharas changelog is from 2009 with version 9. The timestamp for the date modified from is 2002 and the Kithara DLL-files also indicates that it is from 2002 with the string 'Copyright 1996-2002'.

How it is Used
Kithara is the last line of loaded drivers in the boot log at `Windows\ntblog.txt` on the C: drive. There are drivers named the Kithara base driver, I/O accelerator,

Hardware toolkit and Timer toolkit. The file `WINDOWS\zzz_anderung_rs232_neu\0 RPT_aend_nach_inst_rs232.HTM` contains an *'Installation Report: (two-phase mode)'* from 2009, with overview of added and deleted files and registry edits. It is generated by InCtrl5 which is an installation logger for Windows. One of the files changed is the base driver for Kithara. There is an installer file `ksetup.exe` and an executable that apparently loads Kithara drivers named `kloader.exe` in the folder `WINDOWS\system32`. The folder is also where the drivers `kbase6.dll`, `kioa6.dll`, `khdw6.dll` and `ktmr6.dll` reside.

Information Gathering
The Kithara web page `http://kithara.de` can be viewed as it was presented in 2002 using the *The Internet Archive*, a free service providing access to digital artifacts. The Internet Archive also confirms that version 6 was available in 2002.

The functions of the base driver running on the programmer were extracted by reverse engineering using IDA Pro, and are listed in Appendix B. A description of the Kithara modules installed is also there. Besides, we retrieved a handbook for developing with Kithara version 7.

Adobe Acrobat Reader

The programmer is running Adobe Acrobat Reader version 5.0.

Registry keys for Adobe Acrobat Reader are located in `HKEY_LOCAL_MACHINE\SOFTWARE\Adobe\Acrobat Reader\5.0` and `HKEY_LOCAL_MACHINE\SOFTWARE\Adobe\Acrobat Reader\5.0\AdobeViewer`. The registry key `TrustedMode` is set to True(1). TrustedMode is a mechanism to prevent malicious PDF documents from launching arbitrary executable files or write to system directories or the Windows Registry.

However, this version of Adobe Acrobat Reader has 27 known vulnerabilities, where 19 are rated as critical [Cora]. All critical vulnerabilities require no authentication, allows remote access and involves low or medium complexity. 13 allows arbitrary code execution, five might allow arbitrary code execution, and one has an unknown impact.

Also, eight available Metasploit modules exploit a subset of the critical vulnerabilities.

Chilkat

Chilkat is the software used to compress and encrypt patient data. It uses ChilkatZip2.dll version 12.1.1.0, which has a vulnerability rated medium severity according to CVSS v3.0 that allows remote attackers to create or overwrite arbitrary files. This could then be leveraged for code execution. There exists an exploit in Metasploit, for a newer version of the ChilkatZip2.dll library, that has been tested on Windows XP Professional with Service Pack 2 [Corb]. Testing the exploit code on the virtual programmer resulted in the rewriting of a chosen file, confirming it also works on this build of Windows XP Embedded.

Renesas

Renesas is a manufacturer of microcontrollers and semiconductors. There are entries in the registry at location
`HKEY_LOCAL_MACHINE\SOFTWARE\Renesas\Renesas Flash Development Toolkit`
with references to version 4.03.001 which was released in 2008. Searching for it under the documentation section at the Renesas home page lists a PDF with the title
`Flash Development Toolkit Revised to V 4.03 Release 01` which matches the version number from the registry entry.

The description of Flash Development Toolkit reads: *'Renesas Flash Development Toolkit is the dedicated flash programming software for Renesas flash microcomputers, which offers sophisticated and easy-to-use Graphical User Interface'* [Cord].

There are hints that Renesas is used to connect the programmer to a Renesas Flash device customized by Biotronik.

The readme file in
`D:\Program Files\Renesas\FDT4.03\kernels\ProtB\7144\Renesas\1_0_00`
reads:

```
SH/7144F, 7145F
Kernel Version 1.0.00.000

This kernel is built by the following tool-chain.

 SH SERIES C/C++ Compiler Ver. 6.0A
 SH SERIES C/C++ Standard Library Generator Ver. 1.0A
 SH SERIES CROSS ASSEMBLER Ver. 5.0B
 Renesas Optimizing Linkage Editor Ver. 7.0
```

SH SERIES seems to reference the SuperH RISC engine family, which is a reduced instruction set implemented by microcontrollers and microprocessors for embedded systems. Under 'Tools Administration' and 'Communication Tools' it is possible to generate an HTML-file of tool information with much detailed information. The HTML-file shows that the path variable `EGS_root` is connected to the folder `D:\Bio`. We recognize `EgsMain.exe` as the primary software used by the programmer, and the Bio-folder to contain software and tools specific to Biotronik.

Other clues underpin this that Renesas is used to connect the programmer to a Renesas Flash Device.

1. The Flash Development Toolkit documentation reveals that it supports serial and USB communication.
2. The file `TD.exe` has the option *'connect to device'*.
3. Renesas has a baud rate setting in the registry.
4. Under "Tools Administration" and "Communication Tools" there is an E1USBDrv.Dll version 3.00.00.00, GenericSerial.DLL version 1.6.0.0.

The location of the Renesas Flash device is unknown. However, the Programming Head (PGH) is connected to a special 14-pin port depicted in figure 4.9 with a cable labeled with USB, which links it to the clues mentioned above. However, the use of commands from `ftd2xx.dll` belonging to Renesas in the Biotronik testing tool `attrib_edit.exe` described below underpins that it can be used to change the firmware of a pacemaker device. As such, the Renesas Flash Device might be on the pacemaker itself.

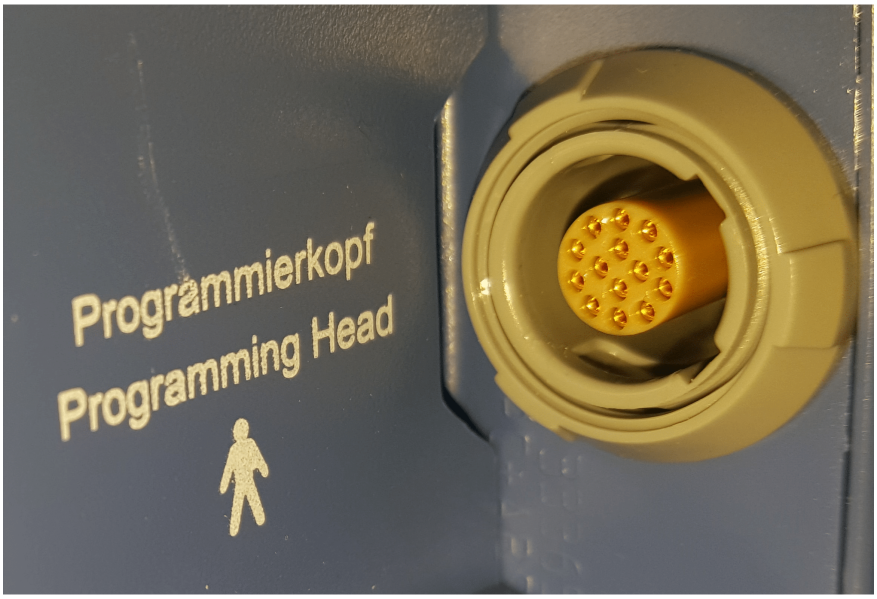


Figure 4.9: 14-pin port for Redel P series adapter to connect the programming head

Anycom

The programmer is using a Bluetooth driver from ANYCOM with version 2.0.0.26. The ICS 3000 technical manual describes an accessory Bluetooth USB flash drive which can be connected to printers compatible with a generic HP driver for wireless printing. Using the *The Internet Archive* we can see from ANYCOMs homepage in 2004 that the CF-2001 Printer Card which is supported by the Bluetooth driver is a recommended Bluetooth adapter by Hewlett-Packard. The readme-file in `D:\Program Files\ANYCOM\ANYCOM Blue Card` states that the driver supports LAN Access and Dial-Up Networking profiles in the Bluetooth 1.1 specification and hardware such as the CF-2001 printer card.

Internet Explorer

`drivercheck.exe` lists an entry for Internet Explorer referencing the file shdocvw.dll of version 6.0.2900.2180. The file is the DLL for Internet Explorer which has 209 known vulnerabilities, where 171 are rated as critical [Cora]. These could simplify exploitation by leveraging Internet Explorer vulnerabilities to achieve arbitrary code execution.

MS XML

Microsoft XML Core Services is a set of services that allow applications written in JScript, VBScript, and Microsoft development tools to build Windows-native XML-based applications. The programmer has version 4.20.9818.0 installed, which has nine known vulnerabilities, where four are rated as critical and allows remote attackers to execute arbitrary code [Corc].

Furthermore, MS XML support for SP 2 expired in April 2010 [Micb].

VC7 Runtime lib

Msvcr71.dll, Mscvr71d.dll, and Atl71.dll are distributed with an application built with Microsoft Visual C++ .NET 2003 with the Microsoft .NET Framework 1.1, and dates back to 2003. This indicates that software is built with .NET Framework 1.1. However, this does not mean that the programmer has the vulnerabilities in the .NET framework, and there are to our knowledge no known vulnerabilities in these runtime DLL-files with version 7.10.3077.0.

Pdf955

Pdf955 is a software to create PDF files and is likely used to create PDFs when exporting data from the programmer. The file `pdfsave.exe` is run when exporting patient data as PDF, which can be viewed with a process monitor.

tshark

Tshark is the touch driver for the screen and is referenced in InstImageCD.log.

Windows XP Components

One interesting file is the `w100b-sp2-Drive C.log` in the `C:\Windows` folder, which lists all components included when building the original Windows XP Embedded image in 2006. These components are possible to cross-check with vulnerability databases to find vulnerabilities in Windows XP. However, without being connected to the network, our attack surface of interest is the USB interface targeting the import function in `EgsMain.exe`. Examining vulnerabilities in Windows XP components could explain the gap between the number of vulnerabilities found by us and Whitescope in what we believe is a Biotronik programmer. However, this would be time-consuming and we therefore decided not to investigate this further to prioritize other interesting paths.

Summary of Vulnerabilities in Commercial Software

The following table summarizes known vulnerabilities in commercial software found on the programmer. The most severe vulnerabilities in the Common Vulnerability Scoring System (CVSS) v3.0 standard are rated critical, with a vulnerability score from nine to ten out of ten. This imply a high score on metrics such as impact on CIA, the remediation level of a solution, and if user interaction is needed for exploitation.

Table 4.1: Summary of vulnerabilities found in commercial software.

Software	**Version**	**Vulnerabilities**	**Critical vulnerabilities**
Adobe Acrobat Reader	5.0	27	19
Chilkat Zip ActiveX control	12.1.1.0	1	0
Renesas	4.03	0	0
Anycom	2.0.0.26	0	0
Internet Explorer	6	209	171
MS XML	4.20.9818.0	9	4
Windows XP Embedded	SP2	Not investigated	Not investigated
Total		**246**	**194**

4.2.4 Proprietary Software

Also present on the programmer at `D:\Bio\Execute\Tools` is what looks to be a folder with internal testing tools. All of these tools do not seem to be mandatory for having the programmer perform its diagnostic features. As such, they are interesting binaries that may tell us more about the development environment of the programmer.

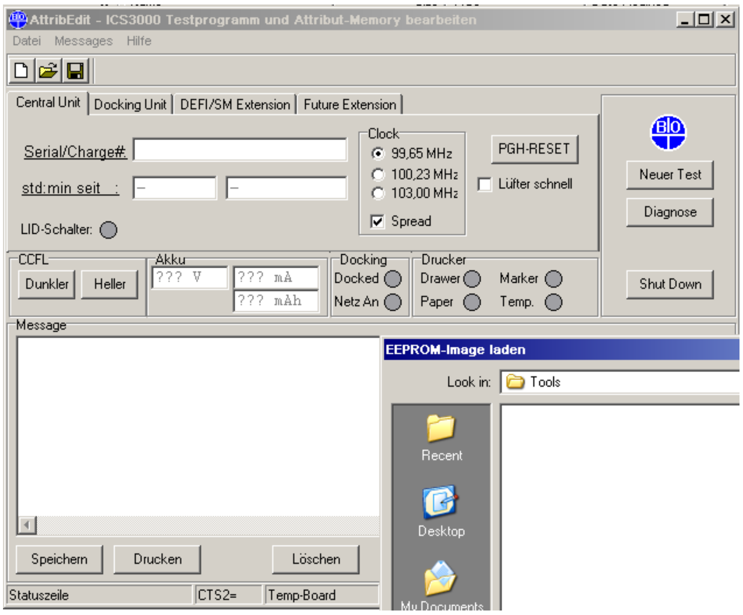
PGH

PGH seems to be the software used by the proprietary Programming Head (PGH). The PCB inside the PGH has has `PGH2000` engraved.

AttribEdit

While the buttons and the program interface is in German, it is capable of editing attributes. More specifically it seems from the title of the program it has the capability of editing memory attributes of pacemakers or their firmware. If one clicks the 'open file' dialog one is asked for an EEPROM file, most likely the ones that are flashed onto the pacemakers. The executable includes a subset of method calls in the file `ftd2xx.dll`, which belongs to Renesas discussed above.

Most likely this is an internal testing tool for pacemaker firmware updates. Furthermore, it can be argued that it is simply present on the programmer for the reason that it is convenient for a Biotronik engineer who knows how the system works, instead of having a dedicated debug unit which they may have to carry around.



Clock: [✓ 99,65 MHz] 100,23 MHz 103,00 MHz
[✓ Spread]

Figure 4.10: Attribute Editor running on our VM.

Bremse

Bremse, meaning 'brake' in German, is quite confusing at first sight. When running, it seems to occupy 100% of the CPU indicated by the performance graph of process explorer in red, meaning it is CPU usage in kernel mode. If one pulls the scrollbar to the right, the graph turns green instead. Green color means the sum of kernel and user mode execution in process explorer. Judging from this basic dynamic analysis, we can guess this is a CPU stress testing tool.

DevListenProject

DevListenProject is not very useful to run, as it most likely is a tool for listening to devices, which could be pacemakers, or maybe the PGH. Without being able to run this program inside a proper programmer, we can only guess based on the static analysis. It may look like it listens for telemetry data, which could be telemetry sent back and forth between pacemakers and programmers in order to verify correct functionality.

DIS_Test

Since the program crashes while being run inside the VM, judging from the static analysis, the acronym DIS_Test stands for Device Interface System Tester, meaning yet another debug tool for device testing. There are also numerous references to the PGH. Finally, there are also strings inside the binary that indicate functionality, such as
`Select one of the object listed below to initialize the PGH device:`. Another interesting path to attack the wireless RF protocol could be to reverse engineer these binaries responsible for testing, instead of a black-box approach. This is further elaborated in chapter 6.

DriverCheck

As briefly mentioned, this program lists and checks for software versions being their correct version, or whether certain system parameters exists, like registry keys. As such it is a handy tool to document what tools are in use or not for our programmer, and especially version numbers.

IcsDebugLog

Additionally to these testing tools, they have a dedicated testing tool for `EgsMain.exe`, which as mentioned before is the application that acts as the main software running on the programmer. It logs exceptions, what libraries are loaded as well as processes and threads spawned. It stores the log file at `D:\Bio\Execute\Log\ICSDebugLog.log`.

InstImageCD

While not entirely clear in its purpose, judging from the log file it also leaves inside the `Log` folder, it seems to be related to updating software via CD. If executed by us in the VM, the log file returns with ‘No CD-code entered!’, implying one needs a special code to be able to update the programmer before the next entry says ‘Cancelled by user’.

UseCheckUtil

Relating back to DriverCheck, it seems this tool also does some checks. More specifically it has a help text when running without parameters, so the functionality here might be the actual functions used in DriverCheck which performs the checks that produce the list, as this program seem to provide us with system details regarding version numbers. Both this and DriverCheck use the file called `control.bcu` which has the format for the different function calls the program supports. Overall, they produce the same error messages in different formats, as such one could argue this possibly is a wrapper or older version of DriverCheck, or the other way around.

4.2.5 Pacemaker Programmer Issues

In this section, we present issues which aim to answer **RQ.1** by addressing security protection mechanisms as well as artifacts which relate to the security model described in subsection 3.1.1.

Authentication

In most cases, computers have some form of authentication in place. Our pacemaker programmer does not have any form of authentication mechanism implemented, meaning anyone with access to the device has full control to perform arbitrary actions. Without any authentication in place, it becomes impossible to provide non-repudiation or accountability.

Windows XP Embedded

Our programmer was purchased from eBay in late 2015. Specifically, we have a version of Biotronik’s programmer called ‘ICS 3000’. A newer release, called ‘Renamic’ exists as well. Its operating system, as depicted in the previous section is Windows XP Embedded with Service Pack 2, which in itself is a security risk due to it being old with known vulnerabilities and out of support.

Table 4.2: Lifecycle deadlines downloaded from https://support.microsoft.com/en-us/lifecycle/search on 11th of may 2018.

Products Released	**Lifecycle Start Date**	**Mainstream Support End Date**	**Extended Support End Date**	**Service Pack Support End Date**
Microsoft Windows XP Embedded Service Pack 1	10/22/2002	Not Applicable	Not Applicable	4/10/2007
Microsoft Windows XP Embedded Service Pack 2	1/18/2005	Not Applicable	Not Applicable	1/11/2011

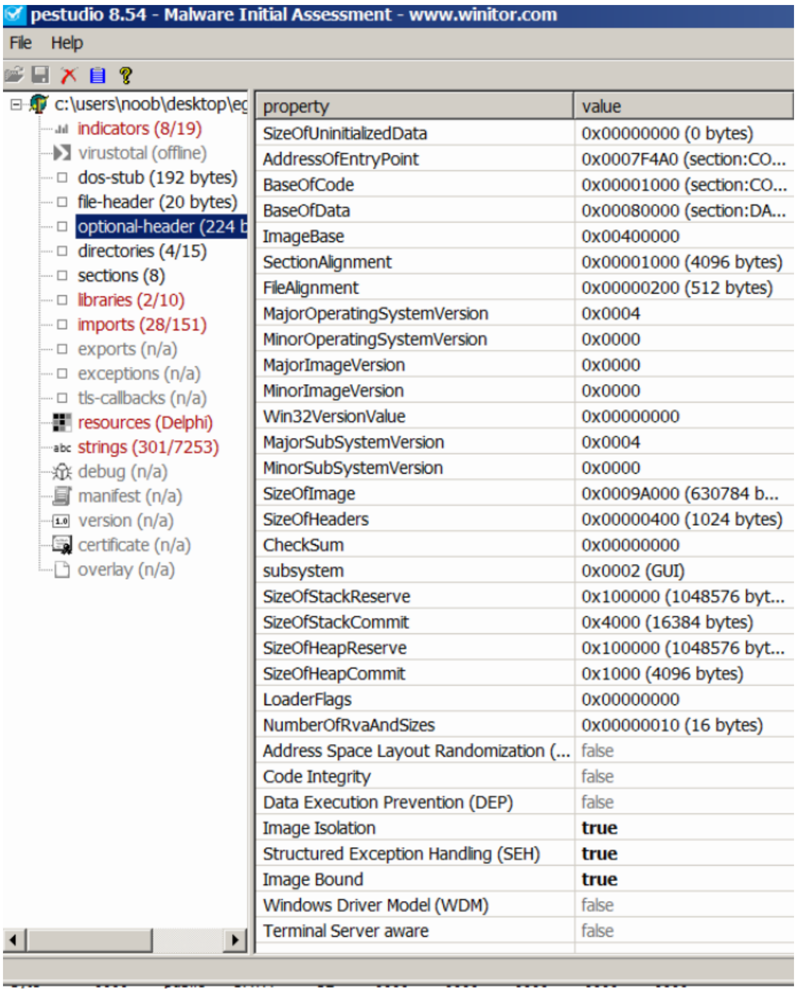
Furthermore, during our visit to a cardiovascular clinic in Norway, we witnessed the startup procedure of their Renamic programmer. Judging from the behavior of the boot procedure it also seemed to have Windows XP Embedded installed. Even if it would be the latest service pack, their extended support for the Windows Embedded platform stopped on 12th January 2016 [Mas], meaning the Renamic programmer would also be outside this window and at risk of having known vulnerabilities that will not be patched by Microsoft.

System and Binary Security Measures

For all computer systems, there are certain measures one can make to prevent or substantially increase the difficulty of vulnerabilities from being exploited even if they exist in the current software version. Firstly, there is Address Space Layout Randomization (ASLR). ASLR helps prevent attackers from predicting addresses, effectively either making sure the attack needs to leak memory addresses somehow or brute force it, if the operating system is not randomizing with enough entropy, or the address space is too small, i.e., 32-bit, which is the case here. Another problem with Windows XP is that it may or may not support system-wide ASLR at all. Followingly, binaries themselves can also have ASLR enabled. Among other security features, this value can be read from the file header. Further randomization is possible with Position Independent Code (PIC). Instead of the program having its executable parts at a fixed base address, which also can be found in the headers of the binary, it will be random too. Thirdly, one must also strive not to allow execution of data located in the data section of the program. In Windows, this is done by employing Data Execution Prevention (DEP).

In our programmer, we can use tools to check some security features of the system such as DEP, while others can be checked with tools directly on the binaries. For checking the binary itself, we used ‘pestudio’ [Win], and for being able to look at the system security properties on the running VM, we used Microsoft SysInternals Process Explorer [Mice].

Evident by the screenshots below, there is no ‘ASLR’ tab in the properties page for the `EgsMain.exe`, meaning it is disabled for this system. Furthermore, you can see the status of ‘DEP’ being disabled. Also, in the screenshot from pestudio it states the file ignores DEP and ASLR, has no integrity checks and has a static ImageBase address, meaning PIC is not enabled either.

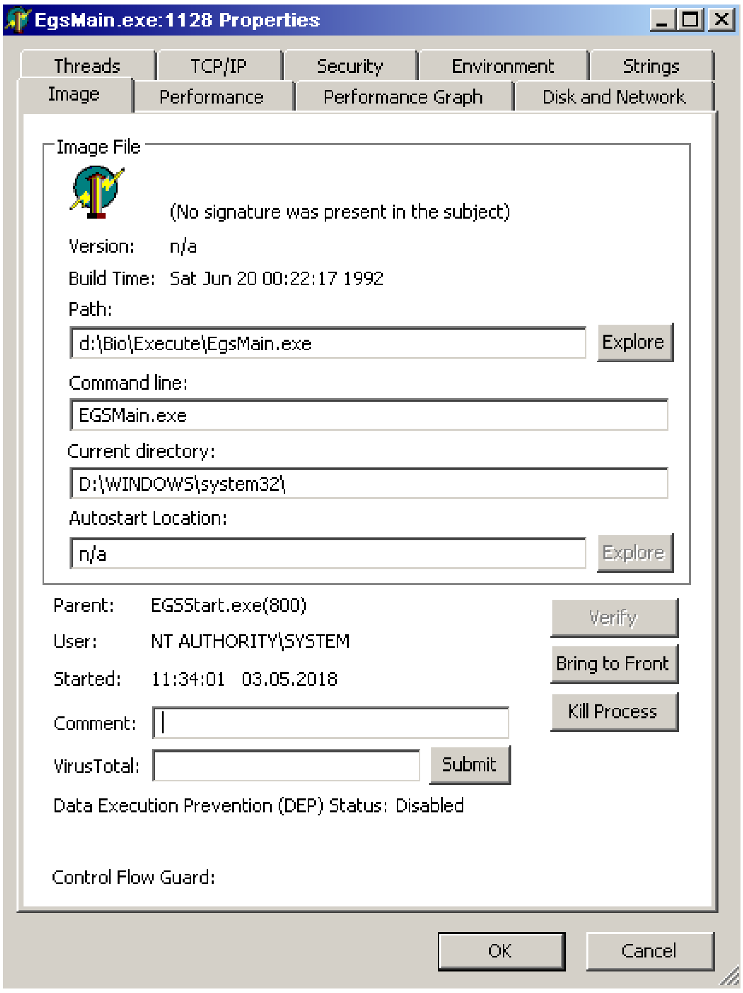

pestudio 8.54 - Malware Initial Assessment - www.winitor.com

File Help

- c:\users\noob\desktop\eg
  - indicators (8/19)
  - virustotal (offline)
  - dos-stub (192 bytes)
  - file-header (20 bytes)
  - optional-header (224 b
  - directories (4/15)
  - sections (8)
  - libraries (2/10)
  - imports (28/151)
  - exports (n/a)
  - exceptions (n/a)
  - tls-callbacks (n/a)
  - resources (Delphi)
  - strings (301/7253)
  - debug (n/a)
  - manifest (n/a)
  - version (n/a)
  - certificate (n/a)
  - overlay (n/a)

property	value
SizeOfUninitializedData	0x00000000 (0 bytes)
AddressOfEntryPoint	0x0007F4A0 (section:CO...
BaseOfCode	0x00001000 (section:CO...
BaseOfData	0x00080000 (section:DA...
ImageBase	0x00400000
SectionAlignment	0x00001000 (4096 bytes)
FileAlignment	0x00000200 (512 bytes)
MajorOperatingSystemVersion	0x0004
MinorOperatingSystemVersion	0x0000
MajorImageVersion	0x0000
MinorImageVersion	0x0000
Win32VersionValue	0x00000000
MajorSubSystemVersion	0x0004
MinorSubSystemVersion	0x0000
SizeOfImage	0x0009A000 (630784 b...
SizeOfHeaders	0x00000400 (1024 bytes)
CheckSum	0x00000000
subsystem	0x0002 (GUI)
SizeOfStackReserve	0x100000 (1048576 byt...
SizeOfStackCommit	0x4000 (16384 bytes)
SizeOfHeapReserve	0x100000 (1048576 byt...
SizeOfHeapCommit	0x1000 (4096 bytes)
LoaderFlags	0x00000000
NumberOfRvaAndSizes	0x00000010 (16 bytes)
Address Space Layout Randomization (...	false
Code Integrity	false
Data Execution Prevention (DEP)	false
Image Isolation	**true**
Structured Exception Handling (SEH)	**true**
Image Bound	**true**
Windows Driver Model (WDM)	false
Terminal Server aware	false

Figure 4.11: pestudio showing the header information of EgsMain.exe.



(No signature was present in the subject)

Version: n/a

Build Time: Sat Jun 20 00:22:17 1992

Path: d:\Bio\Execute\EgsMain.exe

Command line: EGSMain.exe

Current directory: D:\WINDOWS\system32\

Autostart Location: n/a

Parent: EGSStart.exe(800)

User: NT AUTHORITY\SYSTEM

Started: 11:34:01 03.05.2018

Comment:

VirusTotal:

Data Execution Prevention (DEP) Status: Disabled

Control Flow Guard:

Figure 4.12: SysInternals Process Explorer properties page, without ASLR tab and showing DEP being disabled.

Conclusively the system does not have any of these protection mechanisms in place, meaning once someone potentially has found a vulnerability in any software running on the machine, exploitation could potentially be trivial compared to modern systems where these mechanisms are enabled.

Hard Drive Encryption

As mentioned in both MedSec, Bishop Fox and Whitescope, hard drives for both Abbott Medical and the four unnamed vendors were not encrypted [RB17, p. 17] [Liv16, p. 27]. This is also the case with our Biotronik programmer. Evident by the disk image, we can identify the partitions of each drive, and we are never prompted for a password input when we mount it. As a result, we are able to create our VM without going through the hurdle of breaking disk encryption or dumping the disk of a running machine which can lead to an incomplete state and corrupted data. Not only does it ease the creation of our VM, but it also means we can identify the files on the disk image in great detail, whether or not they are signed or if confidential information is stored securely, as documented further below. Such a process would be a lot more difficult if the programmer also uses some form of internal storage compared to a removable medium such as a 3.5” hard drive.

```
ICS3k.001: DOS/MBR boot sector MS-MBR 9M german at offset 0x10+0xFF "
    Ung\201ltige Partitionstabelle" at offset 0x12b "Fehler beim Laden
    des Betriebssystems" at offset 0x151 "Betriebssystem fehlt", disk
            signature 0x6bd9de3f; partition 1 : ID=0x7, active, start-
    CHS (0x0,1,1), end-CHS (0x3ff,254,63), startsector 63, 36869112
    sectors; partition 2 : ID=0x7, start-CHS (0x3ff,0,1), end-CHS (0
    x3ff,254,63),           startsector 36869175, 41254920 sectors
```

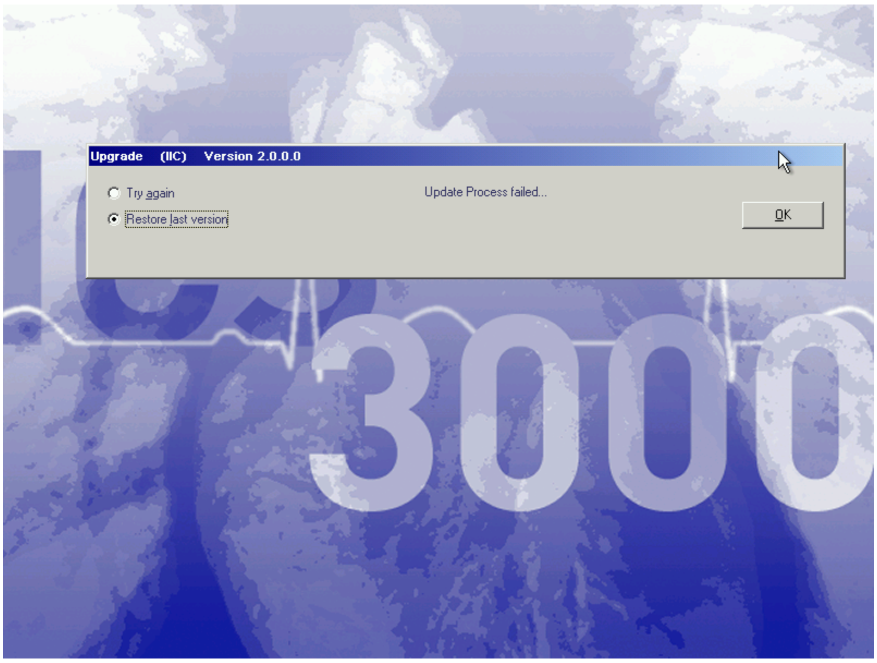
Listing 4.1: Output of the file command on a Linux system, presenting us the boot sector info in German.

```
Disk ICS3k.001: 37.3 GiB, 40007761920 bytes, 78140160 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x6bd9de3f

Device        Boot    Start      End  Sectors  Size Id Type
ICS3k.001p1 *            63 36869174 36869112 17.6G  7 HPFS/NTFS/exFAT
ICS3k.001p2        36869175 78124094 41254920 19.7G  7 HPFS/NTFS/exFAT
```

Listing 4.2: Output of the fdisk -l command on the ICS3k.001 disk image, reading both partitions as NTFS.

Each partition has a specific purpose. When the system was left untouched, it boots to the second partition, which contains the Bio folder with all the software for the programmer functionality. The first partition, which we were able to boot to via changing the boot.ini file seems to be some sort of recovery partition in case a software update fails or similar. We are presented with the following prompt asking us to revert to a previous software version or retry the one we have allegedly attempted when we boot to the C: partition.



Update Process failed...

Figure 4.13: Error box when booting to the C: partition after editing boot.ini.

Another point to make is that when trying to understanding the difference between the two partitions, the fact both were bootable caused confusion. For clarity, if one wishes to edit the registry or other files, be sure to mount the proper partition as mentioned when creating the VM, namely partition 2, sda2, which is called the D: drive in Windows.

Anti-Reverse Engineering Efforts

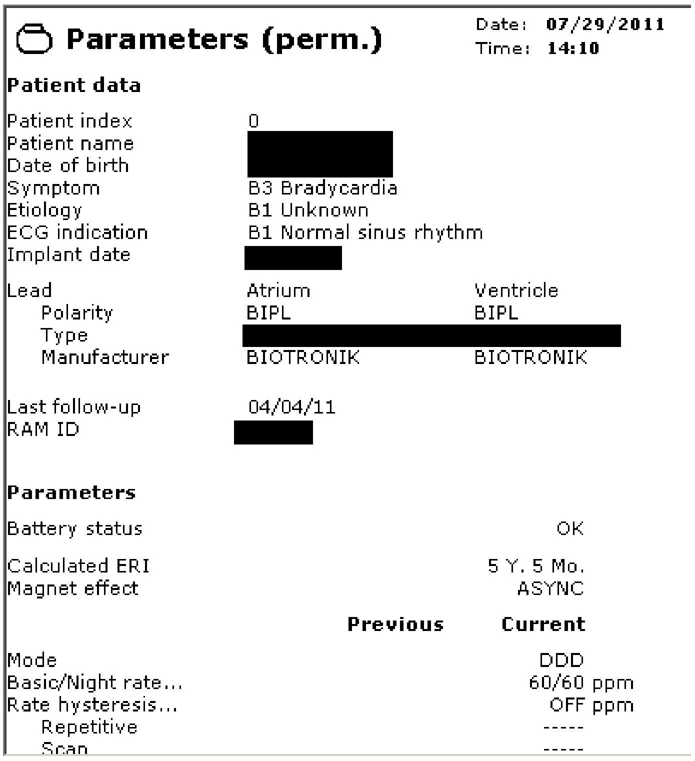
While the libraries are compiled and stripped of their function names, they are straightforward to open and provide little to no effort in attempting to make sure the code is not fairly readable for a reverse engineer. Although the programmer software and its third-party and commercial libraries require considerable time and effort to figure out of due to the sheer size, it could still be made more difficult than just having a complex system. By doing this, one essentially slows the adversaries process of reverse engineering. We found no debug files from the alleged C++ project even though we did find a string indicating the usage of a PDB file for the project of certain files inside the binary, such as the `EGSStart.exe` file, which we believe is responsible for starting the programmer software [Micc]. There are many other pdb files too, revealing their path and environment of the original project. For instance, searching for these pdb files and inspecting their folder structure allows us to gain a tiny picture of said project development environment.

```
for i in \$(find Documents/Infosik/pacehacking/Bio); do strings -f -a \
    $i 2>/dev/null | grep -v BioCommon | grep "pdb\$" | grep "EgsV01";
    done | cut -d " " -f2 | sort
```

Listing 4.3: The output can be found in Appendix C

Patient Data Encryption

Whenever the programmer interrogates a pacemaker, one receives the data stored on the pacemaker to diagnose the patient, and either append to the already existing patient data or adds it as a new entry in case it is their first interaction with this particular programmer. As such, patient health information is stored permanently on the programmer. All patient data are stored unencrypted on the hard drive in a folder called `D:\Bio\Execute\Printmandata` either as an XML, PNG (which are exported from graphs), or RAM dumps from pacemakers themselves. Patient data should not be stored unencrypted and should require some form of authentication to allow physicians to access the data only when needed. Overall this presents a risk of patient data confidentiality being compromised should an attacker get a hold of the machine physically.



Parameters (perm.)

Date: 07/29/2011
Time: 14:10

Patient data

Patient index	0	
Patient name	[redacted]	
Date of birth		
Symptom	B3 Bradycardia	
Etiology	B1 Unknown	
ECG indication	B1 Normal sinus rhythm	
Implant date	[redacted]	
Lead	Atrium	Ventricle
Polarity	BIPL	BIPL
Type	[redacted]	
Manufacturer	BIOTRONIK	BIOTRONIK
Last follow-up	04/04/11	
RAM ID	[redacted]	

Parameters

Battery status		OK
Calculated ERI		5 Y. 5 Mo.
Magnet effect		ASYNC
	Previous	**Current**
Mode		DDD
Basic/Night rate...		60/60 ppm
Rate hysteresis...		OFF ppm
Repetitive		-----
Scan		-----

Figure 4.14: Example of patient data which is presented to us in the programmer, stored unencrypted on the disk.

Patient Data Import and Export

One of the interesting functions of the programmer is the fact that one has the ability to export and import patient data through the data manager interface. One can import data from special ZIP files, or export them to either PDF or ZIP. These special ZIP files do not conform to the ZIP file standards. Since this is one of the few attack vectors where we could potentially get a payload into the programmer using a USB stick, our steps on how we figured out what made these ZIPs are documented in this chapter. The importance of the structure and understanding of these ZIP files are critical to continue work in order to search for potential vulnerabilities. By using a disassembler called IDA Pro coupled with knowledge of how the format of a ZIP file is exported, we were able to deduce how they were created inside the programmer software.

In addition, exporting patient data is not a feature which requires explicit authentication, meaning if anyone has access to the machine for a short amount of time there is a possibility to exfiltrate patient data.

ChilkatSoft ZIP Libaries Firstly we were informed by previous members of the project of some interesting library files on the programmer itself, called names similar to `ChilkatZip.dll`. These libraries were the ones containing the actual functionality for encrypting a ZIP file. Reading the documentation on the ChilkatSoft websites, we understand the ChilkatZip2 file is the main library responsible for setting up the context when creating such a file [Sofe]. A simple string search for references to this DLL should match, in most cases, which files that either depends on or utilize this library and could lead us to the area where the files are encrypted and its routine. At the time, nobody had confirmed these files were indeed encrypted by Chilkat. Since the library was so old, if one were to try to identify if the ZIP files indeed were encrypted or encoded by a current version of the library, one would get an error message stating this ZIP is not encrypted nor password protected, which is wrong. The example code from their website even has a comment stating it should be able to read the encryption property and confirm if it indeed is of an earlier format too, but the check does not work for our ZIP [Sofc, Sofa].

```
anders@pacehacker \$ python DetectEncryptionChilkat.py
This zip is NOT password-protected.
This zip is not encrypted.
Success!
Amount of files in zip: 294
```

Listing 4.4: Output of the python script with an up-to-date Chilkat library claiming our ZIP file is not encrypted at all. Python source is in Appendix D

Following our investigation this is now confirmed, it does utilize the older library version.

```
anders@pacehacker \$ grep -R ChilkatZip2

Binary file Shared_Tlb/ChilkatZip2.tlb matches
Shared_Tlb/MK_Tlb.log:tlibimp7 mit ChilkatZip2.tlb
Binary file ChilkatZip2.dll matches
Data/coffbase.txt:ChilkatZip245dd00 00
Binary file Hcc.dll matches
MK_Tlb.log:tlibimp7 mit ChilkatZip2.dll
MK_Tlb.log:tlibimp7 mit ChilkatZip2.dll
MK_Tlb.log:tlibimp7 mit ChilkatZip2.dll
MK_Tlb.log:tlibimp7 mit ChilkatZip2.dll
MK_Tlb.log:tlibimp7 mit ChilkatZip2.dll
Binary file Tools/attrib_edit.exe matches
```

Listing 4.5: String search for references to the library.

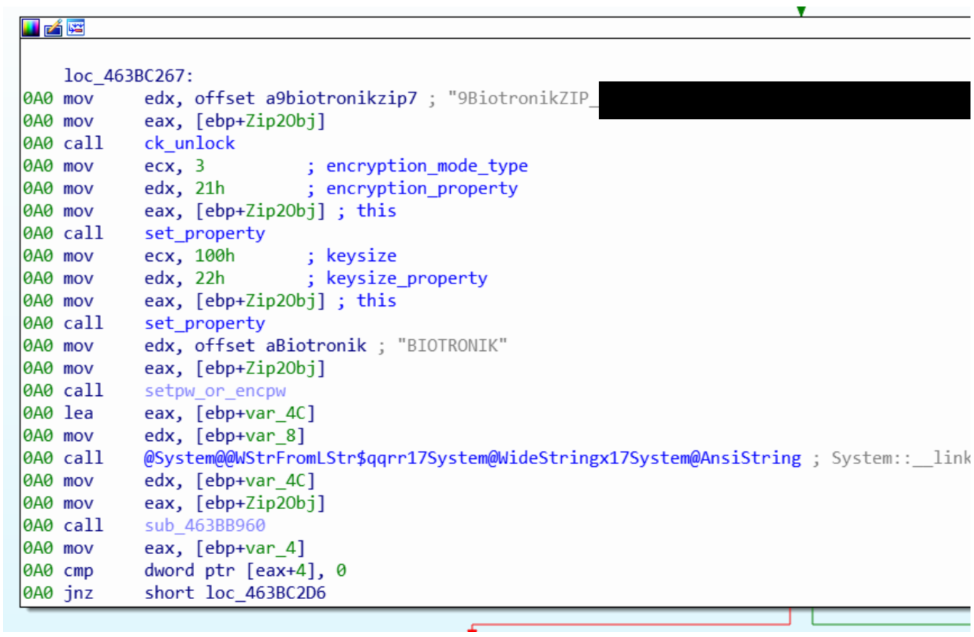
We then chose to investigate these libraries further and the single executable. While `attrib_edit.exe` did not seem to actively use functionality from the Chilkat library, `Hcc.dll` did. It created several Chilkat library context objects in which it used the methods from the Chilkat library. Another string search for `chilkat` returned interesting strings related to TLB files. A description of how TLB files can be used in the Chilkat library is availble on Chilkat forums [Sofd]. As mentioned in subsection 4.2.3 the version of the library has vulnerabilities dated to 2008. Since the webpage describing how was last edited in 2007, we can assume the same functionality still applies.

```
IChilkatZip2
ChilkatZip2Lib_TLB\
TChilkatZip2UnzipPercentDone
TChilkatZip2WriteZipPercentDone
TChilkatZip2FileUnzipped
TChilkatZip2ToBeAdded
TChilkatZip2FileAdded
TChilkatZip2ToBeZipped
TChilkatZip2FileZipped
TChilkatZip2ToBeUnzipped
TChilkatZip2
TChilkatZip2
ChilkatZip2Lib_TLB#
ChilkatZip2Lib_TLB
```

Listing 4.6: The result of running strings -a Execute/Hcc.dll| grep -i chilkat

Investigating the usage of the Chilkat libraries As such we opened IDA Pro which has the ability to identify where inside the file `hcc.dll` the particular strings are referenced. The most interesting one is `TChilkatZip2`, as it provided us with the most interesting subroutines. In particular, subroutine `sub_463BC1A4`. In here there is a smaller portion of code named `loc_463BC267`. After reading the documentation of Chilkat and having written a short script on how to interact with the library as a developer would do, looking at the source code for a demo script to decrypt a ZIP and the particular subroutine in question, some striking similarities arise. Particularly there are at least three string arguments in common going into different function calls:

1. Chilkat license activation string
2. Filename
3. Password in a potentially encoded form

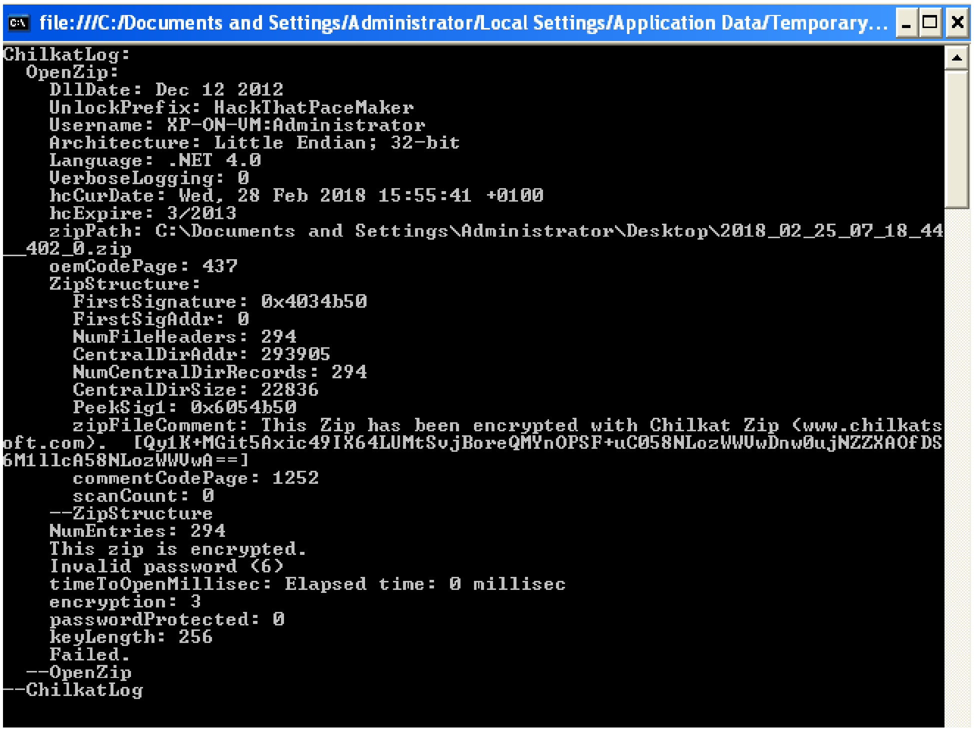


```
    loc_463BC267:
0A0 mov     edx, offset a9biotronikzip7 ; "9BiotronikZIP_[redacted]
0A0 mov     eax, [ebp+Zip2Obj]
0A0 call    ck_unlock
0A0 mov     ecx, 3          ; encryption_mode_type
0A0 mov     edx, 21h        ; encryption_property
0A0 mov     eax, [ebp+Zip2Obj] ; this
0A0 call    set_property
0A0 mov     ecx, 100h       ; keysize
0A0 mov     edx, 22h        ; keysize_property
0A0 mov     eax, [ebp+Zip2Obj] ; this
0A0 call    set_property
0A0 mov     edx, offset aBiotronik ; "BIOTRONIK"
0A0 mov     eax, [ebp+Zip2Obj]
0A0 call    setpw_or_encpw
0A0 lea     eax, [ebp+var_4C]
0A0 mov     edx, [ebp+var_8]
0A0 call    @System@@WStrFromLStr$qqrr17System@WideStringx17System@AnsiString ; System::__link
0A0 mov     edx, [ebp+var_4C]
0A0 mov     eax, [ebp+Zip2Obj]
0A0 call    sub_463BB960
0A0 mov     eax, [ebp+var_4]
0A0 cmp     dword ptr [eax+4], 0
0A0 jnz     short loc_463BC2D6
```

Figure 4.15: Portion of sub_463BC1A4 with the relevant function calls in assembly.

From the figure, we can see the Biotronik license string being put into what would be the "unlock" feature of the Chilkat library. Furthermore, now that we might guess this is the encryption stub we are looking for, we know certain parameters need to be set, such as encryption mode and key length. Finally, before creating the encrypted ZIP file you have to specify a password. Evidently, even though key length and mode of encryption were chosen securely to AES and 256 bits, the password was chosen to be `BIOTRONIK` and then most likely expanded to 32 bytes, as is normal for string passwords in symmetric encryption using a key derivation function. Since we do not possess the Chilkat source code, we cannot confirm if it does indeed use a function like PBKDF2. However, they do have PBKDF2 implemented in their Crypt2 library, and it is therefore reasonable to assume that they may use it for their ZIP library as well. This is not needed to be confirmed though, as long as the method is implemented the same way for the old Chilkat libraries for other programming languages.

Armed with this information, we can now recreate the code given we find a library that is old enough to support the legacy decryption routine. Browsing the internet, it was not a simple task to find proper old library files. We tried to contact Chilkat themselves, but they did not respond to either a forum post or email to customer support. After ending up finding the library at [Sofb], we could now reconstruct the code to decrypt all patient data found on the programmer, as well as encrypt arbitrary data to use as input for our attack vector. We sucessfully managed to decrypt data exported from our programmer. The proof-of-concept code to decrypt and encrypt can be found in the Appendix E. In case the old .NET Chilkat library disappears off the internet it can also be found in the project's repository.


file:///C:/Documents and Settings/Administrator/Local Settings/Application Data/Temporary...

```
ChilkatLog:
  OpenZip:
    DllDate: Dec 12 2012
    UnlockPrefix: HackThatPaceMaker
    Username: XP-ON-VM:Administrator
    Architecture: Little Endian; 32-bit
    Language: .NET 4.0
    VerboseLogging: 0
    hcCurDate: Wed, 28 Feb 2018 15:55:41 +0100
    hcExpire: 3/2013
    zipPath: C:\Documents and Settings\Administrator\Desktop\2018_02_25_07_18_44
__402_0.zip
    oemCodePage: 437
    ZipStructure:
      FirstSignature: 0x4034b50
      FirstSigAddr: 0
      NumFileHeaders: 294
      CentralDirAddr: 293905
      NumCentralDirRecords: 294
      CentralDirSize: 22836
      PeekSig1: 0x6054b50
      zipFileComment: This Zip has been encrypted with Chilkat Zip (www.chilkats
oft.com).  [Qy1K+MGit5Axic49IX64LUMtSvjBoreQMYnOPSF+uC058NLozWWVwDnw0ujNZZXAOfDS
6M1llcA58NLozWWVwA==]
      commentCodePage: 1252
      scanCount: 0
    --ZipStructure
    NumEntries: 294
    This zip is encrypted.
    Invalid password (6)
    timeToOpenMillisec: Elapsed time: 0 millisec
    encryption: 3
    passwordProtected: 0
    keyLength: 256
    Failed.
  --OpenZip
--ChilkatLog
```

Figure 4.16: Output of the C sharp script in Appendix E. Pay attention to the encryption mode and key length, lining up with the assembly code in Figure 4.15. It also confirms the ZIP is encrypted, contrary to what the python script told us earlier.

Potential to Compromise Patient Confidentiality We now asked ourselves the question, 'Can we decrypt exported data from any Biotronik programmer?' We reasoned the answer had to be yes, seeing as pacemakers and programmers are devices in need of thorough testing to ensure they are safe. Meaning they also stay in an active environment for a long time since releasing a new one would take a long period of time. While not proven generally, we had exclusive access to data retrieved from a different programmer than our own ICS 3000, namely a Renamic programmer belonging to a hospital in the Netherlands. Using our new-made scripts we were able to decrypt the patient data from the Renamic programmer as well.

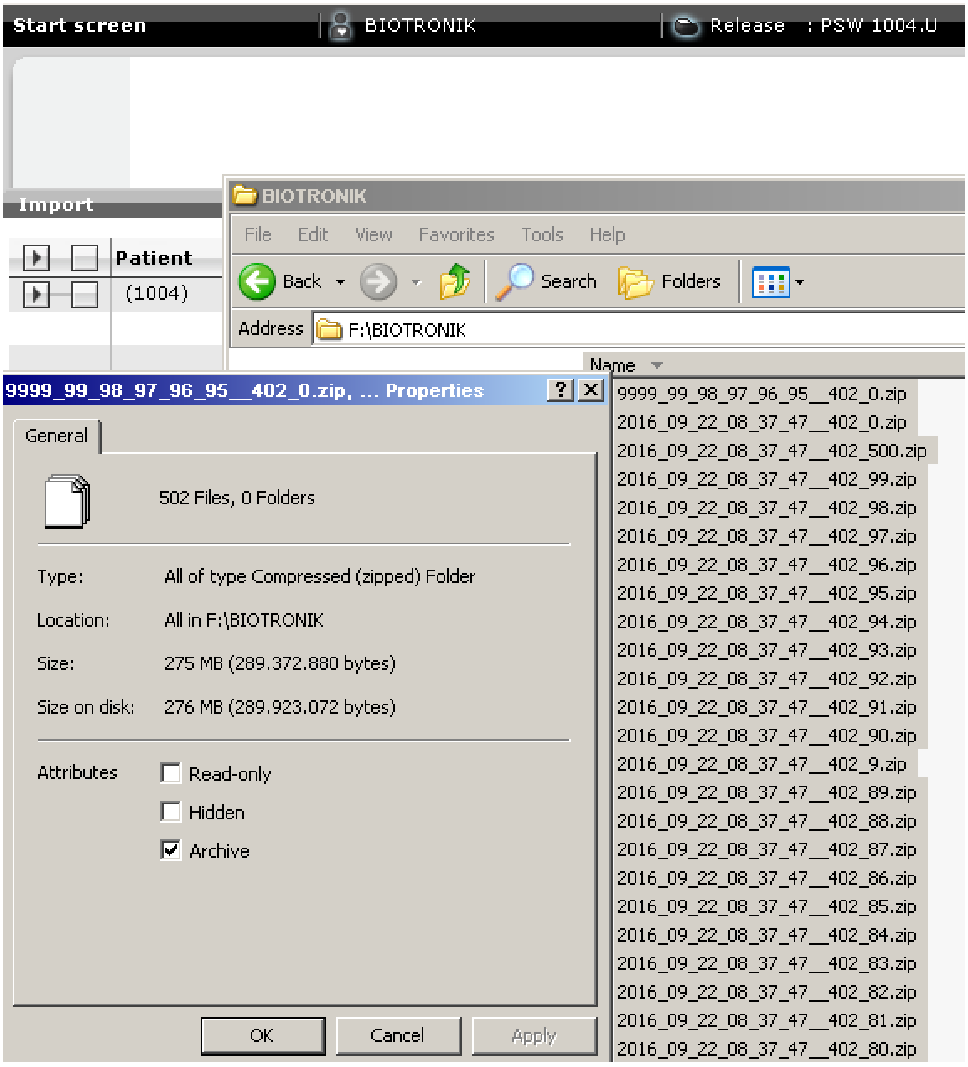
It would appear then, as if all Biotronik programmers have this password hard-coded for the process of exporting and importing ZIP data. This is bad for a number of reasons. Firstly, using a static password means there is no key revocation mechanism in place for existing programmers. This fact also makes it difficult for Biotronik to support legacy data that has already been exported. Secondly, this password enables us to create our own input data which the programmer will attempt to parse. This is an entrance for a potential fuzzer. Thirdly, this discovery could result in a sophisticated attacker being able to exfiltrate data from any Biotronik programmer in the world. Consequently, the hard-coded password enables an attacker to compromise patient data confidentiality.

4.2.6 ZIP Fuzzer

Mentioned before, the ZIP data that is exported and imported to the programmer is one of our potential attack vectors. As such we would like to create a tool to fuzz these libraries that handle the ZIP data to search for vulnerabilities which can exploit the box and let us run arbitrary code.

Time permitting, we would have written a fuzzer to try to find new vulnerabilities in the ZIP file format handler. Biotronik's name format whitelisting served its purpose in slowing down analysis. Nevertheless, we were able to defeat it as described below.

ZIP Name Format One of the first hurdles to understand how the programmer reads the file is the very specific date format filename. In itself, this can also be fuzzed, but if we are to provide seemingly reasonable files for the programmer to believe to be valid in order to pass the name check, one has to figure out the specific filename format requirements. Considering dates are numbers, one of the first simple tests we can try is to see what numbers it accepts. It turns out, the date format string will accept any number from 0–9 for year, month, day, hour, minute and seconds. More specifically, the format turns out to be `Name format: NNNN_NN_NN_NN_NN_NN__402_N.zip , N = [0-9]`. The 402 was mandatory to be there, for unknown reasons so far. These tests were done by terminal commands to copy a legit ZIP file with a normal name, then specifying the different values for all the positions inside the string, then attempting to import them into the programmer. The programmer will then return a number which is twice of the number of accepted files. Important to note is that this does not necessarily mean it does not attempt to read the ZIP file, simply that it returns unsuccessfully.



9999_99_98_97_96_95__402_0.zip, ... Properties

502 Files, 0 Folders

Type: All of type Compressed (zipped) Folder

Location: All in F:\BIOTRONIK

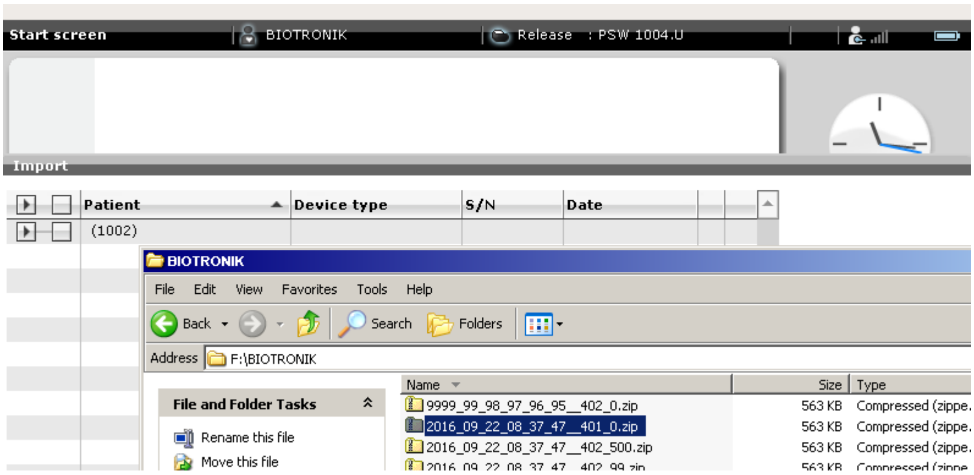
Size: 275 MB (289.372.880 bytes)

Size on disk: 276 MB (289.923.072 bytes)

Attributes
☐ Read-only
☐ Hidden
☑ Archive

```
9999_99_98_97_96_95__402_0.zip
2016_09_22_08_37_47__402_0.zip
2016_09_22_08_37_47__402_500.zip
2016_09_22_08_37_47__402_99.zip
2016_09_22_08_37_47__402_98.zip
2016_09_22_08_37_47__402_97.zip
2016_09_22_08_37_47__402_96.zip
2016_09_22_08_37_47__402_95.zip
2016_09_22_08_37_47__402_94.zip
2016_09_22_08_37_47__402_93.zip
2016_09_22_08_37_47__402_92.zip
2016_09_22_08_37_47__402_91.zip
2016_09_22_08_37_47__402_90.zip
2016_09_22_08_37_47__402_9.zip
2016_09_22_08_37_47__402_89.zip
2016_09_22_08_37_47__402_88.zip
2016_09_22_08_37_47__402_87.zip
2016_09_22_08_37_47__402_86.zip
2016_09_22_08_37_47__402_85.zip
2016_09_22_08_37_47__402_84.zip
2016_09_22_08_37_47__402_83.zip
2016_09_22_08_37_47__402_82.zip
2016_09_22_08_37_47__402_81.zip
2016_09_22_08_37_47__402_80.zip
```

Figure 4.17: Showing 502 files, all with 402 as the name as well as one with a bogus date, accepted and read by the pacemaker indicated by the 1004 number



Name	Size	Type
9999_99_98_97_96_95__402_0.zip	563 KB	Compressed (zippe.
2016_09_22_08_37_47__401_0.zip	563 KB	Compressed (zippe.
2016_09_22_08_37_47__402_500.zip	563 KB	Compressed (zippe.
2016_09_22_08_37_47__402_99.zip	563 KB	Compressed (zippe

Figure 4.18: After renaming one of the 502 files we can see the programmer no more accepts reading the file successfully.

```
for i in seq {1..500}; do cp -v 2016_09_22_08_37_47___402_0.zip 2016
    _09_22_08_37_47___"$i"_0.zip; done
for i in seq {1..500}; do cp -v 2016_09_22_08_37_47___402_0.zip 2016
    _09_22_08_37_47___402_"$i".zip; done
```

Listing 4.7: Bash commands to produce the simple file name test files.

4.3 Findings From Resources Related to the Programmer

Finally we present more artifacts and their context related to the programmer in its ecosystem, which has its origin from elsewhere than the disk image.

4.3.1 Artifacts Retrieved From Exported Data

When decrypting the data files recovered from the programmer in the Netherlands, the files included has lots of logging information that describes the underlying system. In the log files, we found some unique files which were not on our programmer. Firstly, we found an entry for the update code used to update the programmer software to its at the time newest version, 1504.

```
head BioFirst.log
[2015.11.30 12:24:27] Log file created
[2015.11.30 12:24:27] Starting a full installation
[2015.11.30 12:24:27] CMD Param updateCode: 99999528
[2015.11.30 12:24:27] CMD Param singlefilepath: e:\Biotronik_PSW1504A
    -1.exe
[2015.11.30 12:24:27] CountryID: 528
[2015.11.30 12:24:27] Executed "D:\Bio\Execute\Tools\HCCReset2.exe"
[2015.11.30 12:24:27] BuildLabel: PSW 1504.A/1
```

Listing 4.8: Snippet of BioFirst.log from the decrypted data

The entire log has valuable information for a reverse engineer trying to understand the update procedure of the programmer and can be used to explore that path as future work. Here we also get a glimpse of `HCCReset2.exe`'s functionality, which now can be concluded is related to the update procedure. Judging from the other log files as well, we gain valuable information that may help us in the process of understanding the system, presenting an even better reason to properly authenticate the user before exporting data.

4.3.2 Files From Self-Extracting Installer

When extracting the installer files to a directory and running md5sum on both file hierarchies, there are no differences in the file output. We therefore assume these are system files that are the same across system software updates. This could mean, certain software libraries are potentially never patched. Although we cannot say for certain whether or not these files are copied over, it causes concern for what their purpose is, if not to be written to disk. Among these, we find both references to the Kithara shared libraries and Chilkat libraries. Additionally, files which strengthen our belief that the system still runs Windows XP Embedded is also present. Below is a snippet of some of the interesting files found in the two installers, `Biotronik_PSW1504A-1.exe` and `Biotronik_PSW1701A-1.exe`.

```
mkdir 1504unzipped && cd 1504unzipped
7z x Biotronik_PSW1504A-1.exe
for i in $(find); do md5sum $i;done 2>/dev/null > ../1504hashF
cd ..
mkdir 1701unzipped && cd 1701unzipped
7z x Biotronik_PSW1701A-1.exe
for i in $(find); do md5sum $i;done 2>/dev/null > ../1701hashF

diff 1504hashF 1701hashF
1c1
< 46db194f951504b0848a517c7dc78406  ./Biotronik_PSW1504A-1.exe
---
> 01e6a45ee50b1175e9f1e5b93d847555  ./Biotronik_PSW1701A-1.exe
```

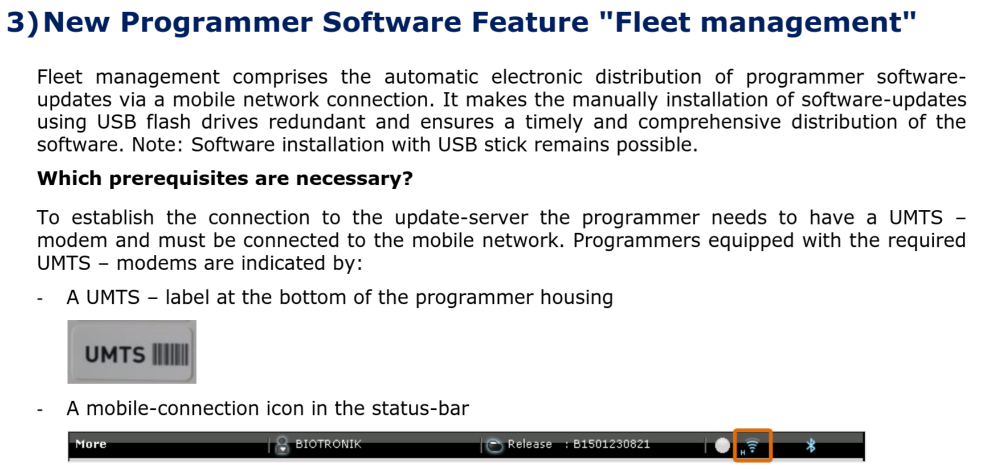
Listing 4.9: Bash commands to create text files with hashes, the output of diff showing only the main binary differs, and none of the extracted files.

```
$ find | grep -i kbas
./W101/WINDOWS/system32/kbas6.dll
./W101/WINDOWS/system32/kbas6.sys
./W100/WINDOWS/system32/kbas6.dll
./W100/WINDOWS/system32/kbas6.sys
$ for i in $(find); do strings -f -a $i 2>/dev/null | grep -i 'chilkat
    '; done
..snip..
./W100/WINDOWS/system32/config/SOFTWARE: ChilkatZip2.ChilkatZipEntry2
..snip..
$ find | grep -i XPe
./W101/WINDOWS/system32/XPePM.dll
./W101/WINDOWS/system32/XPePM.exe
./W100/WINDOWS/system32/XPePM.dll
./W100/WINDOWS/system32/XPePM.exe
$ grep -Ri embedded
..snip..
W100/WINDOWS/system32/eula.txt:Please refer to the End User License
    Agreement that came with your embedded device.
..snip..
#Why have a boot.ini backup file, which isn't used for vista/w7 and
    above? Only XP uses it, and below.
W101/WINDOWS/pss/boot.ini.backup:multi(0)disk(0)rdisk(0)partition(1)\
    WINDOWS="WINXP Embedded Partition 1" /fastdetect /noguiboot /
    bootlogo
W101/WINDOWS/pss/boot.ini.backup:multi(0)disk(0)rdisk(0)partition(2)\
    WINDOWS="WINXP Embedded Partition 2" /fastdetect /noguiboot /
    bootlogo
```

```
..snip..
```

Listing 4.10: Bash commands used to find the various files related to old software and XPe

From the PDF files retrieved from the programmer software, it seems like the newer software and Renamic programmer can update remotely over the internet. This was underpinned by our observation of the symbol `H` in the status bar of the programmer at the cardiovascular clinic, which indicates a mobile connection using *High Speed Packet Access (HSPA)* on a *Universal Mobile Telecommunications System (UMTS)* in accordance with the programmers' user manual.



3) New Programmer Software Feature "Fleet management"

Fleet management comprises the automatic electronic distribution of programmer software-updates via a mobile network connection. It makes the manually installation of software-updates using USB flash drives redundant and ensures a timely and comprehensive distribution of the software. Note: Software installation with USB stick remains possible.

Which prerequisites are necessary?

To establish the connection to the update-server the programmer needs to have a UMTS – modem and must be connected to the mobile network. Programmers equipped with the required UMTS – modems are indicated by:

- A UMTS – label at the bottom of the programmer housing

- A mobile-connection icon in the status-bar

Figure 4.19: Mobile connectivity form user manual for the Renamic programmer with version 1503.A/1

4.4 Findings From Interviews

Below are the responses from the six key informants interviewed, where we focus on their consensus regarding roles at the hospital, the equipment itself, usage of said equipment and access control in place at the hospital. To further gather context for our environment we also ask about typical procedures such as check-ups of patients, and how routines regarding their personal data are handled. The interview guide in Norwegian is available in Appendix G.

4.4.1 Roles at the Hospital

There are in general two roles defined for individuals handling the equipment inside the pacemaker ecosystem, which is doctors and nurses with special training as pacemaker technicians. Our interview subjects consist of both roles, and as such the subjects identify the other staff they interact with on a daily basis as well as head of the clinic.

4.4.2 Selection of Equipment

According to our interview subjects, selection of the pacemaker brand is chosen based on two factors. The first factor is based on a national bidding process, where the outcome is a list of suppliers hospitals can choose from and a percentage distribution of market shares. The second factor is how certain features of a specific brand might perform better related to patient needs, for instance how their sensor technology gives better granularity which in turn gives better information to the doctor. Furthermore, pacemakers are locally stored after being ordered in local storage at the hospital, locked by key card at all times. Programmers, on the other hand, are taken care of by the respective company that has manufactured them, and each hospital might only have a few available to them from each vendor at a time.

4.4.3 Implantation Process

While posing the question of how a pacemaker is implanted, we wished to understand when a programmer is used with the pacemaker for the first time, as well as identifying attack vectors for when a possible timed attack could be most effective. The subjects responded with the pacemaker normally being first interrogated after it is implanted in the patient. From there, rigorous first installment tests are performed, such as checking the values which will be their unique configuration for the condition as well as checking the wires and sensors in the pacemaker itself.

4.4.4 Usage of the Programmer in Its Environment

We then asked how a programmer is used in the hospital. Firstly, there are multiple programmers stored in either the clinic for regular checkups or by operating rooms which are mainly used for the implantation procedure. An important point here is that the programmers themselves are stored in the hallway, often unattended. When asking the follow-up question of where it is stored, some of the subjects responded by theorizing it could very well be possible for anyone to access these machines without anyone noticing. Moreover, whenever these programmers are moved around the hospital, they might be left unattended inside patient rooms by their bed, or if attention is brought elsewhere during a checkup.

4.4.5 Patient Data Deletion

Asking how much data is stored on different programmers revealed that doctors and pacemaker technichians are not aware of any routines to delete data from a programmer. Candidates estimate that somewhere between 50 and a few hundred patients are stored on programmers that support storing patient data, and that this probably includes deceased patients.

4.4.6 Patient Data Flow

In most cases, it could be beneficial to transfer data into the journalling systems in place. These systems do not interact with the programmer, as such our subjects stated they must manually write details into the journalling system or optionally print out information and then scan the printed pages. Exporting data was not common practice for most of our subjects, but they responded with it being a relatively familiar process with USB sticks. Their knowledge about the system and its inner workings ends at 'the data is encrypted'. In addition, the subjects confirmed most programmers do allow to permanently store patient data on them, with the explicit exception of one where it can only be exported by USB stick.
When asked if patients would be given their personal data, the response was in general yes. Followingly, the next question regarding data flow was about the deletion of data. Per protocol, data stored on an old programmer is supposed to be deleted before sent back to the manufacturer and is also transferred by USB stick over to the new programmer.

4.4.7 Internet Connectivity

When asked about whether or not these programmers were connected to the internet, it is very subjective what that might be interpreted as. Some might expect access to the world wide web as internet, while others might realise it can download and upload files regardless of accessing web pages. When queried on the matter, all subjects responded with no, none of the programmers are connected to the internet. This does not match information in the programmer's user manual and our own observation of the programmer.

4.4.8 Patient Safety Concerning the Availability of the Programmer

An important point when considering the security measures we recommend later on, is to have insight as to how critical it may be if all programmers for the specific vendor of the patients' pacemaker is somehow malfunctioning. When asked on the matter, all subjects replied that while there are times where it is more critical to have it function, such as during implantation, even in an emergency situation at the ER the programmer is not a critical component, keeping the patient alive is. Even if a patient has a pacemaker, inside a hospital, the patient will be kept alive. In conclusion, patient health is not at risk, should a programmer stop working or malfunction during an emergency in its normal environment.

4.4.9 Dangerous Configurations of a Pacemaker

Proposing the question, *'Are there configurations which can be dangerous to a patient?'* we wished to uncover whether or not having access to patient data could potentially provide an attacker with enough information to create an exploit which could harm a specific patient. Subjects replied with varied responses, but clarification uncovered that it is possible to turn off emergency shock therapy for ICD patients as well as change threshold values for pacing, effectively rendering a pacemaker useless, since it is not providing enough electricity for the heart to contract.

4.4.10 Decision to Perform Surgery Based on Diagnostic Data

In attempting to uncover the importance of how central the data retrieved from the programmer is, we asked the subjects whether or not the decision to remove cables, one of the more dangerous procedures related to having a pacemaker, could be based solely on the information from the programmer. Overall, the subjects replied with the data coming from the programmer is central in the decision making of removing cables, underlining its importance in diagnostics of potential issues with the pacemaker in comparison to other procedures which are supplementary, i.e., X–Ray or EKG. Cables themselves are only able to be tested by the programmer. The risk associated with such surgery is so significant they have a team of thorax surgeons on standby.

4.4.11 Decision to Apply Cybersecurity Patches

The doctors interviewed explained that certain patients that are fully dependent on their pacemaker are not receiving security patches for their pacemakers, due to the risk involved in the patching process. While the exact number of unpatched pacemakers is uncertain, it indicates potential for improvement in the patching process. Doctors explained how they could perform the update on all patients if the patch was critical, which they didn't consider the cybersecurity update from Abbot to be. The interview subjects also reveal that it might take up to a year for a security patch to be applied in a routine control if the patient is not explicitly called in.

4.4.12 Patient Safety Routines

When considering consequenes of deliberate pacemaker misconfiguration and unavailable programmers, it is apparent that the cardiovascular clinic has resources and procedures to ensure patient safety. In an acute situation, the focus is on keeping the hearth pumping regardless of the pacemaker. Additionly, the problem of a misconfigured pacemaker can be resolved by using a strong magnet.

4.5 Summary

To summarize our findings, we have a working platform for which the research project can continue with the VM as it creates a common ground for all participants. In addition, we have pointed out several inadequacies, such as an unencrypted hard drive and no authentication mechanism to use the programmer itself.

Furthermore, we have proven for two different programmers that we can decrypt patient data exported from them, violating patient data confidentiality. Our testing of this vulnerability implies exporting data with a static password is not specific to a programmer model, and indicates it can be possible to decrypt data from any Biotronik programmer in the world.

Finally we have gathered insights of how changes in context can affect people and procedures by interviewing key informants with expert knowledge of the pacemaker ecosystem.

These findings enable attack vectors with associated impact on key principles in information security listed in Table 4.3. A separation is made between exporting data, and decrypting exported data, because the programmer can export unencrypted PDF files in addition to encrypted ZIP files. Physical access to the programmer also breaches the principle of confidentiality by allowing a user to observe patient data without authentication.

Table 4.3: Attack vectors from results.

Attack vectors	**Impact on key principles**
Exploiting vulnerabilties in the OS	Confidentiality, Integrity and Availability
Exploiting vulnerabilties in commercial software	Confidentiality, Integrity and Availability
Data export using the USB interface	Confidentiality
Decrypting exported data	Confidentiality
Data import using the USB interface	Integrity and Availability
Physical access to the programmer	Confidentiality
Physical theft	Confidentiality and Availability

Chapter 5
Countermeasures

In this chapter, we answer **RO.3** by describing countermeasures to mitigate attack vectors documented in chapter 4. Additionally, we propose a new programmer adopting countermeasures described in this chapter.

5.1 Software

5.1.1 Operating System

Whenever one designs special software for systems responsible for life-critical functions, it is important to consider the use of operating systems and third-party or commercial libraries with great care. Effectively, using a modern operating system increases the security level of the ecosystem. Integrity is more difficult to compromise with the advantage of an up-to-date system with fewer known vulnerabilities. Successful exploitation of certain known vulnerabilities can affect all three principles of the CIA model, making it very important to stay up-to-date on the patching procedure for both the operating system and its software.

Firstly, by using Windows XP Embedded after it has reached end of support implies that the operating system has known vulnerabilities that can be exploited to compromise the integrity of the system. Also, as observed during a visit to a cardiovascular clinic, the Renamic programmer most likely runs Windows XP Embedded in 2018. Even if it uses service pack 3, it is no longer supported by Windows [Mas].

Secondly, there are challenges with using Windows XP Embedded in a closed environment. Considering the options for updating this particular system is either done through a CD or a USB stick, patching the operating system itself requires an in-person visit. Hence, updating these systems require a person with knowledge of the system, such as a Biotronik engineer or certified technician. As for our programmer, it is then what we call ‘air-gapped’. An air-gapped system is one which is not currently connected to the same network as other machines (to the best knowledge of the staff), and updates have to be delivered out of band. While one may believe these computers are not connected to the internet, it is difficult to say if this is the case or not. From the PDF files retrieved from the programmer software in subsection 4.3.2, it seems like the newer software and Renamic programmer can update remotely over the internet, and that connectivity is enabled by default. Such a case can provide a false sense of security, where one believes that a system is not connected to the internet when it, in reality, is connected [Lev11]. All key informants were of the opinion that the programmer is not connected to the internet, despite our findings in the user manual and observation of the programmer in use at the cardiovascular clinic.

Therefore, due to the difficulty of providing a closed environment reliant on commercial software, we would recommend using a real-time operating system instead. These operating systems are tested based on their behavior at failure time, ensuring the stability of the system is the top priority. A downside to using such an operating system, especially if one develops it independently, is its time-consuming manner of testing. As such, given this is unreasonable to require from a medical device compared to, i.e., an air traffic controller system, one could still utilize specialized versions of commercial operating systems.

Both Linux and Windows provide these, although it requires vendors to update their systems from Windows XP to for instance Windows 10 to harden their systems. As previously shown, Windows XP Embedded might have received system updates until 2016, but application-level security measures were not adequate, minimizing the effort required to potentially exploit a device if a vulnerability is discovered. Of course, there are Linux distributions available as well for such purposes, but the most important point here is to ensure security updates are regularly and easily applied to the systems.

In the same manner, protection mechanisms such as ASLR and DEP must be turned on and supported by the operating system. Most modern systems today apart from certain specific Embedded systems have this enabled, and would certainly not be a problem on such a machine that uses a commercial operating system.

Binary Security Protection Mechanisms

Binaries should also be compiled using security protection mechanisms to mitigate the risk of exploitation of a vulnerability. PIC, binary specific ASLR, code integrity checks, and buffer overflow protection mechanisms such as the `/GS` switch for Visual Studio [Micd] are examples of security protection mechanisms that should be enabled.

5.1.2 Commercial Software

Similar to operating systems, commercial software also has an update process for both security and functionality updates. Most critical to medical devices would be to avoid breaking changes, while balancing security needs. Our observation that the process of exporting and importing patient data rely on a very old commercial library is concerning. Nowadays, both Linux and Windows provide tools internally to secure storage of files adequate for the security requirements of patient data. More specifically, the CryptoAPI by Microsoft or the Crypto API for Linux which are both implemented directly at kernel level [Mica, MV]. Hence there is no need to use an at least ten-year-old Chilkat library for securely exporting and importing patient data.

5.1.3 Cold Storage Encryption

Encrypting data at rest is an important step in ensuring confidentiality of sensitive data. Inside the pacemaker ecosystem, there are patient data stored on every device. Therefore, each of them needs to employ a secure and tamper-resistant storage module. For pacemakers, a more significant focus on the hardware security could be applied due to its energy efficiency so that power is not spent on encryption. Instead, gluing components together and creating a much more restrictive environment for hardware hackers is most likely a better approach. Below is a picture of a pacemaker where the chips are available after desoldering the outer layers.

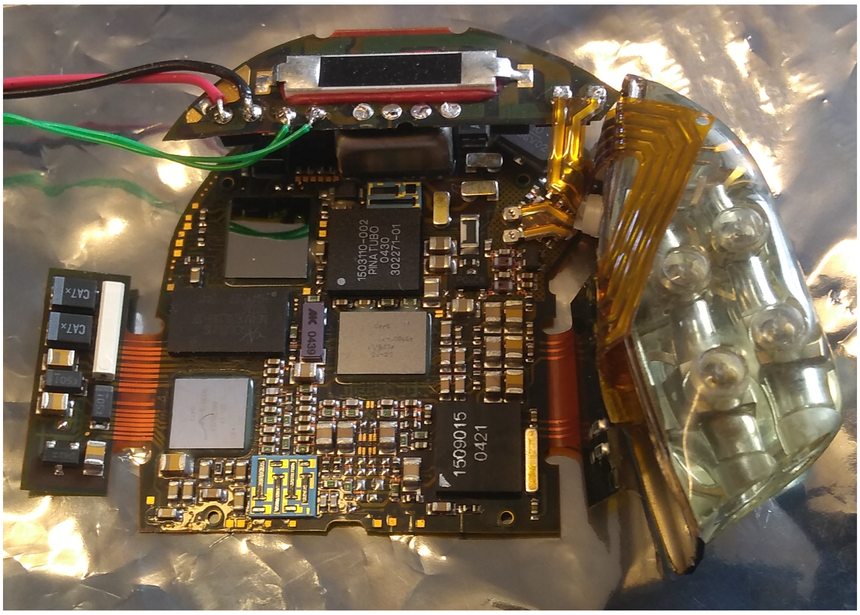


Figure 5.1: Pacemaker with outer components desoldered giving access to the underlying chips.

As for the programmer, we recommend disk encryption. Depending on the choice of OS this can be implemented in several ways. On the Windows platform, it is possible to deploy BitLocker such that only authenticated users in an Active Directory (AD) can access the disk. Combining this with the key access cards already in use in hospitals today should provide an adequate security level without significantly worsening the availability of the machine for normal usage in its regular environment. Adversaries attempting to steal the machine would then also need a valid set of credentials to decrypt the disk.

Additionally, patient data should be encrypted and decrypted only when needed to be displayed in the programmer. Whenever the programmer is active and in use, all patient data except for the patient currently being diagnosed should be stored encrypted and out of reach from someone who could potentially gain access remotely.

5.1.4 Anti-Reverse Engineering Efforts

Software

Anti-reverse engineering efforts are effective against reverse engineers by complicating the process of understanding the system. Examples of these efforts are encrypting functions, packing the code and stripping them of function names and symbols. Although these efforts may slow down the execution time of the binaries slightly, it should not affect the program while running. Hence, its performance impact should be negligible and at the same time improve the time complexity until a possible vulnerability would be discovered. For this, we do not wish to provide an example, as most solutions are commercially sold. Another important factor to consider is that these solutions may introduce new problems, and their robustness might not be guaranteed concerning availability. Therefore we propose it as a suggestion and encouragement for vendors to apply, but strictly not necessary.

Hardware

For all three units inside the ecosystem, we also recommend concealing or disabling the debug ports such as UART and JTAG, making it considerably more difficult for an adversary to connect to the board with privileged access. Mentioned before, the manufacturers have not spent enough effort on concealing their debug interfaces.

5.2 Hardware

5.2.1 Storage

Currently, the programmer uses removable commercial hard drives in 3.5“ format, which is easily removable and easy to analyze with commercial tools. Therefore, we propose to embed the storage device into the motherboard or PCB where critical files and software can be stored securely and tamper resistant. This is not strictly needed, as full disk encryption might be a simpler and cheaper alternative. Regardless, depending on the vendors' requirements, a dedicated secure storage module might be a better solution for some vendors. It should be underlined that either this or full disk encryption are two different ways of achieving the desired outcome, a more tamper resistant environment. Ideally, both would be preferred, both a physical tamper-resistant storage module that employed full disk encryption. The confidentiality of the disk then rests upon the security of the passphrase or keys stored on smart cards authorized to decrypt the disk.

5.3 Authentication

Authentication, as mentioned in section 4.2.5, was nonexistent on the programmer device for both interrogating with a pacemaker and to access the system and its sensitive data. With GDPR now in effect, it is mandatory that patient data comply with special requirements for processing, and that sensitive information follows requirements for storage and retrieval [oEU16, p. 121]. Even though subsystems analyzed by Whitescope and MedSec did not have any form of patient data encryption, we have observed encrypted patient data exported from the data manager. Therefore, for all manufacturers, it is necessary to implement authentication to ensure patient data confidentiality. Additionally, patient data should be easily available and not require unnecessary hassle to access for pacemaker technicians. Below are situations where we believe it should be required to implement an authentication mechanism:

- Booting the programmer itself to decrypt the hard drive
- Ability to lock and unlock the system in a running state for interrogating or configuring pacemakers
- Accessing patient data
- Exporting patient data
- Importing patient data

For all these situations, the most practical solution which provides an adequate security level for the programmer will be a smart card solution integrated with key access cards. The cardiovascular clinic previously mentioned already use key access cards for authentication on computers. Consequently, this authentication mechanism is familiar to doctors and pacemaker technicians, and is therefore not in conflict with normative constraints. Keys access cards are also possible to synchronize with AD users in corporate Windows environments. Since authentication keys can be revoked and replaced, it makes malicious activities difficult despite physical possession of a programmer. Assuming this authentication mechanism is appropriately implemented, we believe it would be a sufficient step towards a more secure system with the exception of exporting patient data. Due to GDPR and new regulations in the EU on privacy, a second authentication token or passphrase should be used whenever one wants to export or print patient data. For such activities, we suggest employing another physically stored digital key, in the form of a Yubikey or other solutions such as OpenPGP Smart Cards to work as a second form of authentication [Yub, Sho]. Furthermore, the digital key should be stored securely in a locked office, preferably in a safe or similar safe storage device. Even better, would be to also deploy a secure password or PIN on the digital key to strengthen secrecy in case it is stolen.

Authentication can provide stronger non-repudiation and accountability for the programmer by linking the existing logging of actions described in section 4.2.2 with the authenticated user.

Also, while not explicitly looked at by us, the pacemakers should only accept commands received from a programmer when the user is properly authenticated. This may require software updates for the pacemakers, in case the protocol does not already provide such mechanisms. Implementing authentication on the programmer will also increase the security of pacemakers by mitigating the possibility for an adversary to maliciously configure a pacemaker. As a consequence, strengthening the security of the programmer also improves the security of the pacemaker ecosystem.

5.4 Theoretical Proposal

For clarity, we propose a theoretical programmer to ensure that security requirements are met by applying fundamental security measures and using modern up-to-date software and hardware solutions to mitigate vulnerabilities. The suggestion is based on the Windows platform since Biotronik programmers are built on Windows. Proprietary software from Biotronik also has to be updated to be compatible with this proposal. The theoretic programmer has the following specifications.

- Microsoft Windows 10 Long Term Servicing Branch.
- Enabling modern security protection mechanisms such as ASLR, DEP and PIC.
- Updated commercial software.
- BitLocker Drive Encryption.

- Trusted Platform Module for secure storage of private keys.
- Use the CryptoAPI by Windows to authenticate users with key access cards.
- Require authentication in recommended situations.
- Use the CryptoAPI by Windows to encrypt patient data and require authentication for decryption.
- Use the CryptoAPI by Windows to sign exported data and verify the signature of imported data.
- Ensure sensitive data is stored on a secure tamper-resistant chip and encrypted when not in use.
- In place integrity checks of the software running to ensure only untampered software is in use.
- Signed executables to hinder integrity violations of code before executing vital functions. As a part of the authentication scheme, functions that could, if abused, cause patient harm should be encrypted and only decrypted at runtime after authentication is successful.

This proposal aims to improve the development and design of pacemaker ecosystems to achieve a more secure environment. These components, if implemented, provides a more hardened environment for malicious adversaries to operate in compared to the current programmer's environment. The proposed countermeasures are mapped to the attack vectors identified in section 4.5 in Table 5.1.

Table 5.1: Countermeasures to mitigate attack vectors uncovered in results.

Attack vectors	**Impact on key principles**	**Countermeasures**
Exploiting vulnerabilties in the OS	Confidentiality, Integrity and Availability	Windows 10 Long Term Servicing Branch with enabled modern security protection mechanisms and integrity verification of signed executables
Exploiting vulnerabilties in commercial software	Confidentiality, Integrity and Availability	Updated commercial software and integrity verification of signed executables

Data export using the USB interface	Confidentiality	User authentication
Decrypting exported data	Confidentiality	Windows CryptoAPI
Data import using the USB interface	Integrity and Availability	Updated commercial software, integrity verification of signed data using Windows CryptoAPI and integrity verification of signed executables
Physical access to the programmer	Confidentiality	User authentication
Physical theft	Confidentiality and Availability	BitLocker Drive Encryption

Chapter 6 Discussion

In this chapter, we discuss our results in a broader perspective. Validation criteria for the security of programmers are discussed, and our findings of artifacts and their context are evaluated. We discuss connecting the programmer to the internet and debate the availability requirement of programmers. Furthermore, we outline the most significant implications of our research for the people and procedures in the pacemaker ecosystem. Finally, possible attack vectors and attack scenarios are presented before outlining future work.

6.1 Validation Criteria for Security

The security level of the pacemaker ecosystem can be discussed using the security model introduced in subsection 3.1.1. Lysne argues that one must consider what one is trying to protect against to answer if something is secure [Lys15].

What do we need to protect against? The unauthorized configuration of a pacemaker is essential to mitigate. Authorized personnel test the pacing amplitude after the implantation of a pacemaker to optimize power usage while ensuring that the pacing amplitude is above the threshold necessary for the heart to contract. However, setting the pacing amplitude too low for the heart to contract without the resources of a cardiovascular clinic to keep the heart pumping can have a fatal outcome. We also need to mitigate data theft to protect data privacy rights. Besides, integrity must be ensured to protect the trust of the programmer for diagnosis and configuration purposes.

In light of this, we can return to the elements of the security model. Authentication is not implemented, and as such non-repudiation, authenticity and accountability are not provided by the programmer. We argue that authentication should be a security criterion for the programmer to protect data privacy rights and to protect against unauthorized pacemaker configuration. As discussed in section 5.3, we argue for using key access cards for authentication which is already used for authentication in other information systems at the mentioned cardiovascular clinic. Furthermore, we argue for a second authentication factor whenever an action involves accessing patient data or configuring a pacemaker.

Disk encryption and patient data encryption should also be enabled to ensure confidentiality and protect data privacy rights, as previously discussed in subsection 5.1.3. Neither the disk nor patient data are encrypted on the programmer.

Also, the programmer should be hardened to ensure verifiable integrity. This is to protect against unauthorized pacemaker configuration and data theft and to protect the trust of the programmer. The main software used for diagnosis and configuration in the programmer is not signed and is therefore not integrity protected.

Lysne also argues that one must look at the environment, and not only the technical aspects. This underpins why the Design Science Framework by Hevner incorporating both the environment and the technological side is a good fit for this research. A critical part of the environment is the regulations that affect medical devices.

The guidance by FDA and regulation by EU of pre-market and post-market surveillance of cybersecurity in medical devices was discussed in chapter 3.3. This is a continuous process during the lifetime of a device that should detect known vulnerabilities and patch the device to reduce risk without adversely affecting the benefit-risk ratio. However, we have uncovered multiple known critical vulnerabilities in the programmer that have not been patched. These known vulnerabilities would be easy to monitor with publicly available vulnerability data sources such as `https://www.cvedetails.com` given vendors knowledge of commercial and third-party software in their products. This indicates that a post-market surveillance system has not yet been put in place by Biotronik.

The missing patches for known vulnerabilities also indicate that Biotronik is not following best practices in the industry. Not following best practices indicates that Biotronik violates current FDA recommendations and EU regulations. A study of security framework adoption in The USA from 2016 reported that 70 % of the surveyed organizations recognize the NIST Cybersecurity Framework as best practice for computer security [res]. The NIST Cybersecurity Framework is also recommended by FDA in their post-market guidance.

The latest amendment to the EU MDD 2007/47/EC expands the definition of medical devices also to include software, and states that *‘software must be validated according to the state of the art’* [MMC12]. Dr. Martin McHugh describes the state of the art as *‘what is generally accepted as good practice’*, which takes us back to the NIST Cybersecurity Framework.

We suggest that a validation criterion for the security of programmers should be that vendors are collaborating with third-party and commercial software vendors to receive cyber threat intelligence and vulnerability information in accordance with the NIST Cybersecurity Framework. This would enable programmer vendors to apply patches before vulnerabilities become publicly available. It would also allow the post-market surveillance system to be *'actively and systematically'* gathering data following the upcoming EU regulation.

The Joint Research Centre of the European Commission is suggesting that research should be the main driver for vulnerabilities discovery [Pup]. While there is no harmonized standard in the EU for coordinated vulnerability disclosure, we suggest that a validation criterion for pacemaker programmer security is that vendors actively collaborate with researchers to discover vulnerabilities.

Another validation criterion in Norway is if the product is in compliance with the Norwegian Code of Conduct for information security in the healthcare sector. We argue that it is not. The programmer breaches the principle of confidentiality in that *anyone* with physical access could get access to health and personal information. As discussed in chapter 4, data at rest is not encrypted, and there is no authorization of personnel. Moreover, it is not in compliance with integrity, where the Code of Conduct states that security measures are to be taken to stop people or technology from changing information without authorization. As a result of missing security measures for authentication, it is possible for people without authorization to change data. The low adoption of file signatures implies that changes would not be discovered by the programmer itself. Besides, the number of known vulnerabilities simplifies exploitation by increasing the attack surface. Vulnerabilities could be exploited to achieve arbitrary code execution, thereby changing information without authorization.

6.2 Artifacts and Context

As discussed in chapter 2, answering **RQ.1** is a wicked problem. With finite resources and the complexity of the ecosystem, it is not feasible to describe all artifacts and their context. Consequently, we cannot conclude with absolute certainty on the context of artifacts. We argue, however, that our assumptions do give a clearer picture of the security state in the pacemaker ecosystem. As presented in section 4.2.2, we have taken measures to filter out interesting artifacts, where we have focused on artifacts possibly related to **RQ.1**.

One find is that the programmer utilizes old proprietary commercial software. While commonly used software such as Internet Explorer and Adobe Acrobat Reader contain known vulnerabilities, one cannot conclude that less used software is safe because there are no known vulnerabilities. There is a delay from when a vulnerability is introduced and until it is discovered and patched. Without entering the discussion of security in proprietary vs. open source software, it is worth noting that the source code of open source software is more accessible and therefore more available for security testing. While it is not certain whether commercial software vendors perform security tests on their software, it is not transparent if Biotronik does.

Furthermore, we question the necessity of storing vendor specific debug tools on the programmer. While this has helped our understanding of the programmer, it would also aid a potential adversary in the reconnaissance phase of an attack.

6.3 Connecting the Programmer to the Internet

The implications of an air-gapped system was previously mentioned in chapter 4 and chapter 5. On the one hand, connecting the programmer to the hospital network for authentication purposes could provide an even wider attack surface for a network intruder, but on the other hand, having no network connection at all is in general more secure, considering their update procedure is mainly through USB sticks. In its current state, connecting the programmer to the network would be a bad idea. Therefore, it is worrisome that the newer version appears to be connected to the internet as revealed in subsection 4.3.2.

If one decides to implement authentication through smart cards as already mentioned, the security of the programmer is now related to the security of the hospital network. Although it may widen the attack surface if one sufficiently secures the programmer, as described in chapter 5, the criteria for the system security should be sufficient.

6.4 Availability Requirement

While there might be a stricter requirement in general for the availability of medical devices such as a pacemaker, the *programmer* does not need to meet the same level. While we do not propose it needs anywhere near the authentication hardening of a classified network, such as the ones used by governments, having programmers require authentication for accessing functionality is needed. Our interviews indicate there is no immediate need of a programmer in an emergency situation at the ER, as such it builds a stronger argument for implementing proper authentication for functions such as reprogramming a pacemaker and exporting or printing data.

6.5 Implications for People and Procedures

The programmer is a vital component of the pacemaker ecosystem, as it is used both for the diagnosis and configuration of pacemakers. The diagnosis is an essential basis for the doctor's decisions, and an individual configuration is necessary for patient comfort and to ensure that the pacemaker is functioning. While our interviews indicate that the cardiovascular clinic has resources and procedures to safeguard patient safety in the case of a pacemaker misconfiguration or even without a programmer, these findings can not be generalized to apply for all pacemaker clinics. Also, the safety net of a cardiovascular clinic's resources and procedures are not readily available if an incident occurs elsewhere.

6.6 Possible Attack Vectors

A vulnerability is an instantiation of an attack vector that can be exploited. The sum of attack vectors constitutes the attack surface which can be exploited in an attack scenario. In addition to the attack vectors identified from our results in Table 4.3, our findings and related literature indicate other possible attack vectors.

Whitescope identified UART and JTAG debug ports on an unspecified number of programmers in their study [RB17]. We did not focus on the hardware of the programmer due to the risk of bricking the device, but this could be an attack vector.

The USB interface on the programmer is another possible attack vector we did not test due to the risk of bricking the device. Jodeit & Johns has studied the potential of attacking the USB interface.

> [...] potential attacks are not limited to the USB related code inside the kernel but extend over a large number of different kernel sub-systems and device drivers reachable by USB devices which would not be associated with USB at first glance. The USB protocol allows reaching those parts of the kernel which could otherwise not easily be attacked remotely [JJ10].

Furthermore, the VM could be combined with a PGH to possibly create a functioning programmer. If so, it could be used as an attack vector to interrogate or alter the configuration of a pacemaker. Regardless of the VM, a programmer can also be acquired to interrogate or alter the configuration of a pacemaker. SafeSync is a module to connect a programmer, such as the ICS3000 we have studied, to a pacemaker without using a PGH [Datb]. This functionality could be used to extend the reach of a malicious configuration up to six meters [Datb].

Our finding that newer software supports internet connectivity is also a possible attack vector. Internet connectivity could potentially extend the reach of an attack and scale the possible number of attacks.

The lack of disk encryption and integrity checks implies that changing the hard drive of a programmer is also a possible attack vector.

These unconfirmed but possibly feasible attack vectors are summarized in Table 6.1 which expands Table 4.3 with attack vectors identified in our results.

Table 6.1: Possible attack vectors.

Attack vectors	**Impact on key principles**	**Feasible**
Exploiting vulnerabilties in the OS	Confidentiality, Integrity and Availability	Confirmed
Exploiting vulnerabilties in commercial software	Confidentiality, Integrity and Availability	Confirmed
Data export using the USB interface	Confidentiality	Confirmed
Decrypting exported data	Confidentiality	Confirmed
Data import using the USB interface	Integrity and Availability	Confirmed
Physical access to the programmer	Confidentiality	Confirmed
Physical theft	Confidentiality and Availability	Confirmed
Debug ports	Confidentiality and Integrity	Unconfirmed
USB interface	Confidentiality, Integrity and Availability	Unconfirmed
Internet connectivity	Confidentiality, Integrity and Availability	Unconfirmed
Replace disk	Integrity	Unconfirmed

6.7 Attack Scenarios

In this section, we present three possible attack scenarios based on possible attack vectors identified in Table 6.1. An adversary could perform these attacks with COTS equipment. Thus, an adversary does not need to be very resourceful to perform these attacks.

6.7.1 Malicously Alter System Files

Starting from the disk image that was altered to create the VM, similar techniques could be applied to alter system files with malicious intent. Other possible attack vectors to alter files are vulnerabilities in commercial software, ZIP fuzzing, internet connectivity or the USB interface. Also, we know that the main software `EgsMain.exe`, used by doctors and pacemaker technicians, is not signed. The ability to alter system files could then be used to configure the pacemaker with one value and display another value to the doctors. For instance, the power amplitude ensuring that the heart contracts from pacing could be set lower than the doctor intended to increase patient risk outside the hospital. Diagnosis data for the resistance in leads could be displayed to be artificially high, indicating that the high-risk operation of replacing the leads is necessary.

6.7.2 Maliciously Configure Pacemaker

After acquiring a functional programmer, either using a VM with a PGH or buying a programmer on eBay, there are no authentication mechanisms in place that prevents an adversary from utilizing the same functions as doctors and pacemaker technicians. Consider an adversary targeting a sleeping pacemaker dependent patient by adjusting the pacing amplitude to the point where the heart no longer contracts from pacing. Using the SafeSync module, the attack range could be extended facilitating attacks from nearby seats on buses, trains, and airplanes. Without aid to keep the heart pumping, this attack would have a fatal outcome.

6.7.3 Data Theft

To further expand on how it would be possible for a malicious adversary to gain valuable knowledge about a programmer device, consider the following attack path, focusing on exporting data from the programmer, compromising both patient confidentiality and accessing logs from the programmer itself.

Firstly we need to perform reconnaissance at the hospital to locate the programmer machines. Knowing there most likely will be programmers in the same areas as described in our interviews, the easiest method would be to walk in, as most hospitals are open to the public during business hours. Arguing further, due to the programmers not having authentication, and allegedly being unprotected for minutes at a time during checkups, it is possible for an adversary to walk up to a programmer and export the data to a USB stick. Due to the incomplete security of the encryption key of exported data, an adversary will gain access to logs and all patient data by decrypting them locally. Furthermore, as evident from the results analyzing the data from the programmer from the Netherlands, interesting artifacts about the system and its internal workings can be uncovered as well, i.e., the update code which was found.

6.8 Disclosure Process

We argue by practicing Coordinated Vulnerability Disclosure following ISO/IEC 29147:2014, our thesis has the potential of causing a larger impact on the environment of the ecosystem. If we compare the case against Abbott, which has gotten two security advisories by ICS–CERT, to the Whitescope paper as well as Marin et al. it is unclear whether or not these reports have been followed up. We believe disclosing the details of our analysis through the proper channels in a separate more succinct document will be the correct procedure to cause a meaningful contribution to the pacemaker ecosystem. Hopefully, it will contribute to reducing the risk for patients health and safety, and compromise of patient data.

Dr. Hauser argue that *'[...] patients have a fundamental right to be fully informed when they are exposed to the risk of death no matter how low that risk may be perceived.'* [SOMM10]. As such, we intend to publish this thesis after one year, which should be a sufficient time-frame for Biotronik to handle the Coordinated Vulnerability Disclosure.

6.9 Future Work

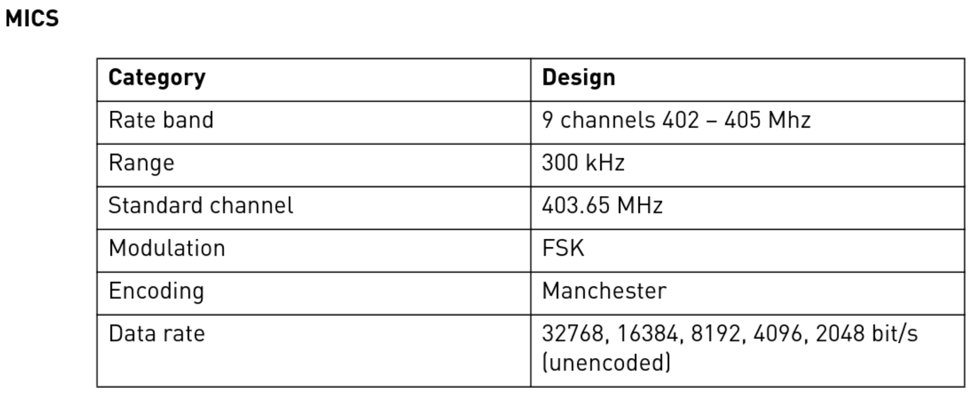
Throughout the project, we managed to open a few possibly interesting doors for new participants to the project. Additionally, other unexplored paths are described here.

6.9.1 Pacemaker Memory Dump

As previously mentioned, we have access to log files including a RAM dump of a pacemaker from the Netherlands. Such log files, as explained, when exported, are zipped encrypted. Since we were able to determine the encryption method and password, we also managed to decrypt the content of this file. Besides, we also have a partial memory dump with many redacted parts. In-depth analysis of this memory dump would be an interesting project path on its own for someone with memory forensics knowledge in embedded devices.

6.9.2 Biotronik pacemaker Radio Frequency Protocol

While reading the documentation for the programmer we came upon a detailed description of how the RF protocol works, which makes the work that must be done simpler for the ones who would like to explore the RF protocol of the programmer by Biotronik.


MICS

Category	Design
Rate band	9 channels 402 – 405 Mhz
Range	300 kHz
Standard channel	403.65 MHz
Modulation	FSK
Encoding	Manchester
Data rate	32768, 16384, 8192, 4096, 2048 bit/s (unencoded)

Figure 6.1: MICS RF details.

At this time of writing there are no publicly available documents analzying the security of the RF protocol used by the renamic programmer.

6.9.3 Home Monitoring Unit Retry

While we initially tried to verify and test previous work done by Whitescope in order to attack the Biotronik HMU, the result leaves the door open for further investigation by others.

6.9.4 Continuing Fuzzing of ZIP Files and Their Contents

There are a couple of things to consider with the fuzzing of the ZIP files which can lead to a potential exploit. Firstly, there are the ZIP files themselves, meaning one would attempt to exploit how the Chilkat library handles its unzip function with bogus data. More clearly, this is an option that does not care for what data it gives to the programmer. It tries to see if the unzip library can or can not handle certain inputs. Another alternative is fuzzing the Biotronik XML files inside the ZIP file, both the patient data and the other log entities [KG]. These files are the ones that are read by the programmer software itself and then displayed to us by the data manager. Fuzzing these files, compared to the ZIP itself means we focus on keeping the ZIP structure intact, and focus on fuzzing the Biotronik XML file format. Any cause for exceptions to be thrown would open up a potential vulnerability which could lead to exploitation of the device. Most preferably we would like the ability to write and execute files.

6.9.5 USB Hardware Interface Fuzzing

While not limited to attempting to exploit the data via a USB stick, one could attempt to exploit the actual USB driver software installed on the programmer. This is a bit more risky, considering the virtual machine uses its own driver to enable USB functionality which is not representative of the real world drivers in use.

Considering this hardware is old, and potentially legacy, the USB drivers used on the physical programmer could be vulnerable to attacks on a hardware driver level. This is different from fuzzing the input data itself, as one attacks the hardware port of the programmer, hopefully causing a malfunction in the system driver for the USB. Since we use a virtual machine for most of our testing, it uses another set of USB drivers in order to enable them for the host machine, as such this fuzzing option is only viable on the physical machine itself, making it quite risky in case we break the expensive programmer.

6.9.6 Third-Party and Commercial Software

As shown with its own tools, namely `drivercheck.exe`, there are several third-party and commercial tools installed which we have not analyzed. Each of them is a new attack vector which could be explored to uncover even more vulnerabilities. Promising candidates are Renesas from 2008 which seems to communicate with a flash device, and Kithara from 2002 with the ability to create code executable at the kernel level. With Kithara, there might be a possibility to inject code at the kernel level by utilizing Kithara drivers or creating an application for Kithara that runs separated from Windows. Kithara also has an optional memory module that enables direct access to physical memory data.

6.9.7 Renamic Programmer

When in possession of a Renamic programmer, verifying the work done in this thesis and testing new functionality from software updates is also a possible path. Most relevant are the new UMTS capabilities for it to have automatic diagnostic data uploaded, a functionality they call 'ReportShare', and an automatic remote update feature. With this enabled it implies the programmer is connected to the internet. The fact we have seen it still may be running Windows XP and is connected to the internet is a cause for concern. These features are described in the PDFs found inside the programmer software update files. These can be found in the project's repository.

Chapter 7 Conclusion

In this research we have set out to answer the following research question:

> *What is the security of the programmer within the framework of the key principles of Information Security: Confidentiality, Integrity, and Availability (CIA).*

We conclude that the security level of the pacemaker ecosystem is inadequate. Shown in Table 4.3, the current implementation of the programmer breaches all three principles of the CIA model. As discussed in chapter 6, the programmer puts the ecosystem as a whole in a more vulnerable state. We have shown that the programmer does not comply with current FDA recommendations, or EU and Norwegian regulations. It contains at least 246 known vulnerabilities where 194 are rated as critical. The programmer has no authentication, and physical access to the programmer is feasible. It uses an OS that has reached end of support, meaning that it no longer receives security updates. Furthermore, it utilizes old proprietary commercial software. Also, we have shown that one can decrypt data from two programmers with different software versions in different countries. This is a breach of patient data confidentiality, and also raises the concern if all exported data from Biotronik programmers are encrypted with the same static password.

As such, we have not been able to disprove our hypothesis that *'The pacemaker programmer has vulnerabilities that make it insecure in its regular environment'*. On the contrary, we have found vulnerabilities that confirms that the programmer is insecure in its regular environment.

References

[Adm] U.S. Food Drug Administration. Developing a software precertification program: A working model. https://www.fda.gov/downloads/MedicalDevices/DigitalHealth/DigitalHealthPreCertProgram/UCM605685.pdf. Accessed: 2018-05-01.

[BG82] Martin L Bariff and Michael J Ginzberg. Mis and the behavioral sciences: research patterns and prescriptions. *ACM SIGMIS Database: the DATABASE for Advances in Information Systems*, 14(1):19–26, 1982.

[Blo16] Carson Block. Mw is short st. jude medical. 2016.

[Bow08] Glenn A Bowen. Naturalistic inquiry and the saturation concept: a research note. *Qualitative research*, 8(1):137–152, 2008.

[C^+11] GH Crossley et al. The clinical evaluation of remote notification to reduce time to clinical decision (connect) trial: The value of remote monitoring. *Journal of the American College of Cardiology*, 57(10):1181–1189, 2011.

[CGH09] Anne Cleven, Philipp Gubler, and Kai M Hüner. Design alternatives for the evaluation of design science research artifacts. In *Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology*, page 19. ACM, 2009.

[Cha11] Kathy Charmaz. Grounded theory methods in social justice research. *The Sage handbook of qualitative research*, 4(1):359–380, 2011.

[Cora] MITRE Corporation. Security vulnerabilities. https://www.cvedetails.com/vulnerability-list.php?vendor_id=53&product_id=497&version_id=10038&page=1&hasexp=0&opdos=0&opec=0&opov=0&opcsrf=0&opgpriv=0&opsqli=0&opxss=0&opdirt=0&opmemc=0&ophttprs=0&opbyp=0&opfileinc=0&opginf=0&cvssscoremin=0&cvssscoremax=0&year=0&month=0&cweid=0&order=1&trc=19&sha=afbcb8f0ddb967791d113f6c0580cb569d1de4f2. Accessed 2018-03-28.

[Corb] MITRE Corporation. Security vulnerabilities. https://www.cvedetails.com/cve/CVE-2008-5002/. Accessed 2018-05-21.

[Corc] MITRE Corporation. Security vulnerabilities. https://www.cvedetails.com/vulnerability-list/vendor_id-26/product_id-1813/version_id-6015/Microsoft-Xml-Core-Services-4.0.html. Accessed 2018-03-28.

[Cord] Renesas Electronics Corporation. Flash development toolkit (programming gui). https://www.renesas.com/en-us/products/software-tools/tools/programmer/flash-development-toolkit-programming-gui.html#documents. Accessed 2018-05-02.

[Cou14] Council of European Union. Council regulation (EU) no 745/2017, 2014. https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:32017R0745&from=EN.

[CWC$^+$08] Jane Chen, Bruce L Wilkoff, Wassim Choucair, Todd J Cohen, George H Crossley, W Ben Johnson, Luc R Mongeon, Gerald A Serwer, and Lou Sherfesee. Design of the pacemaker remote follow-up evaluation and review (prefer) trial to assess the clinical value of the remote pacemaker interrogation in the management of pacemaker patients. *Trials*, 9(1):18, 2008.

[Data] FCC ID Database. 175 khz wireless frequency explorer. https://fccid.io/frequency-explorer.php?lower=0.175&upper=0.175. Accessed: 2018-04-17.

[Datb] FCC ID Database. Safesync safesync module user manual 380184–d-ga_safesyncmodule_en_2011-06-15.fm biotronik se & co. kg. https://fccid.io/QRISAFESYNC/Users-Manual/15-SafeSync-UserMan-1577715. Accessed: 2018-05-19.

[FH13] Robert Farrell and Cliff Hooker. Design, science and wicked problems. *Design Studies*, 34(6):681–705, 2013.

[fHA] Directorate for Health and Social Affairs. Norwegian code of conduct for information security in the health and care sector. Accessed 2018-05-17.

[fTDM] Jonathan Gornall for The Daily Mail. The heart pacemakers at risk from hackers: Sound far-fetched? security experts are treating it deadly seriously. http://www.dailymail.co.uk/health/article-3252609/The-heart-pacemakers-risk-hackers-Sound-far-fetched-Security-experts-treating-deadly-seriously.html. Accessed: 2018-05-10.

[ftNMA] National Executive Committee for the Norwegian Medical Association. Etiske regler for leger. http://legeforeningen.no/Om-Legeforeningen/Organisasjonen/Rad-og-utvalg/Organisasjonspolitiske-utvalg/etikk/etiske-regler-for-leger/. Accessed: 2018-05-16.

[Fu15] Kevin Fu. On the technical debt of medical device security. 2015.

[Gro] IMDRF SaMD Working Group. Software as a medical device (samd): Key definitions. http://www.imdrf.org/docs/imdrf/final/technical/imdrf-tech-131209-samd-key-definitions-140901.docx. Accessed: 2018-05-01.

[HHBR^{+}08] Daniel Halperin, Thomas S Heydt-Benjamin, Benjamin Ransford, Shane S Clark, Benessa Defend, Will Morgan, Kevin Fu, Tadayoshi Kohno, and William H Maisel. Pacemakers and implantable cardiac defibrillators: Software radio attacks and zero-power defenses. In *Security and Privacy, 2008. SP 2008. IEEE Symposium on*, pages 129–142. IEEE, 2008.

[Hir] Ben Hirschler. Gsk and google parent forge $715 million bioelectronic medicines firm. https://www.reuters.com/article/us-gsk-alphabet/gsk-and-google-parent-forge-715-million-bioelectronic-medicines-firm-idUSKCN10C1K8. Accessed: 2018-05-02.

[ICa] ICS-CERT. Advisory (icsma-17-241-01). https://ics-cert.us-cert.gov/advisories/ICSMA-17-241-01. Accessed: 2018-04-25.

[ICb] ICS-CERT. https://ics-cert.us-cert.gov/advisories/icsma-18-107-01. https://ics-cert.us-cert.gov/advisories/ICSMA-18-107-01. Accessed: 2018-04-25.

[Int] Intel. Network adapter drivers for windows xp*, final release. https://downloadcenter.intel.com/download/18717/Network-Adapter-Drivers-for-Windows-XP-Final-Release. Accessed: 2018-04-06.

[ISO14] Information technology – Security techniques – Vulnerability disclosure. Standard, International Organization for Standardization, Geneva, CH, February 2014.

[JJ10] Moritz Jodeit and Martin Johns. Usb device drivers: A stepping stone into your kernel. In *Computer Network Defense (EC2ND), 2010 European Conference on*, pages 46–52. IEEE, 2010.

[Jona] Richard W. Jones. hivexsh(1) - linux man page. https://linux.die.net/man/1/hivexsh. Accessed: 2018-02-02.

[Jonb] Richard W. Jones. libguestfs, library for accessing and modifying vm disk images. http://libguestfs.org/. Accessed: 2018-04-29.

[KG] BIOTRONIK SE Co. KG. Biotronik ieee 11073-10103 xml structure. https://www.biotronik.com/sixcms/media.php/211/biotronik_ieee_11073-10103_xml_structure.pdf. Accessed: 2018-04-29.

[Kro08] David M Kroenke. *Experiencing MIS*. Pearson Prentice Hall,, 2008.

[lab] labgeek. dd2vmdk. https://github.com/labgeek/dd2vmdk. Accessed: 2018-01-29.

[Lee00] A Lee. Systems thinking, design science, and paradigms: Heeding three lessons from the past to resolve three dilemmas in the present to direct a trajectory for future research in the information systems field,“keynote address. In *Eleventh International Conference on Information Management, Taiwan*, 2000. Available at: http://www.people.vcu.edu/~aslee/ICIM-keynote-2000/ICIM-keynote-2000.htm. Accessed: 2018-05-06.

[Lev11] Eireann P Leverett. Quantitatively assessing and visualising industrial system attack surfaces. *University of Cambridge, Darwin College*, 7, 2011.

[Liv16] Carl Livitt. Preliminary expert report of carl d. livitt. 2016.

[LPL$^+$12] Maurizio Landolina, Giovanni B Perego, Maurizio Lunati, Antonio Curnis, Giuseppe Guenzati, Alessandro Vicentini, Gianfranco Parati, Gabriella Borghi, Paolo Zanaboni, Sergio Valsecchi, et al. Remote monitoring reduces healthcare use and improves quality of care in heart failure patients with implantable defibrillatorsclinical perspective: The evolution of management strategies of heart failure patients with implantable defibrillators (evolvo) study. *Circulation*, 125(24):2985–2992, 2012.

[Lys15] Beitland K. Hagen J. Holmgren A. Lunde E. Gjøsteen K. Manne F. Jarbekk E. Nystrøm S. Lysne, O. Digital sårbarhet – sikkert samfunn. 2015. Accessed 2018-05-30.

[Mar96] Martin N Marshall. The key informant technique. *Family practice*, 13:92–97, 1996.

[Mas] Dave Massy. What does the end of support of windows xp mean for windows embedded? https://blogs.msdn.microsoft.com/windows-embedded/2014/02/17/what-does-the-end-of-support-of-windows-xp-mean-for-windows-embedded/. Accessed: 2018-04-15.

[MCPF13] Bryan Marshall, Peter Cardon, Amit Poddar, and Renee Fontenot. Does sample size matter in qualitative research?: A review of qualitative interviews in is research. *Journal of Computer Information Systems*, 54(1):11–22, 2013.

[Mica] Microsoft. About cng. https://msdn.microsoft.com/en-us/library/windows/desktop/aa375276(v=vs.85).aspx. Accessed: 2018-05-04.

[Micb] Microsoft. Msxml 4.0 sp3 release notes. https://www.cvedetails.com/vulnerability-list/vendor_id-26/product_id-1813/version_id-6015/Microsoft-Xml-Core-Services-4.0.html. Accessed 2018-05-12.

[Micc] Microsoft. /pdb (use program database). https://msdn.microsoft.com/en-us/library/kwx19e36.aspx. Accessed: 2018-05-13.

[Micd] Microsoft. We recommend using visual studio 2017 download now /gs (buffer security check). https://msdn.microsoft.com/en-us/library/8dbf701c.aspx. Accessed: 2018-05-01.

[Mice] Microsoft. Windows sysinternals. https://docs.microsoft.com/en-us/sysinternals/. Accessed: 2018-05-13.

[MMC12] Martin McHugh, Fergal McCaffery, and Valentine Casey. Changes to the international regulatory environment. *Journal of Medical Devices*, 6(2):021004, 2012.

[MS95] Salvatore T March and Gerald F Smith. Design and natural science research on information technology. *Decision support systems*, 15(4):251–266, 1995.

[MSG$^+$16] Eduard Marin, Dave Singelée, Flavio D Garcia, Tom Chothia, Rik Willems, and Bart Preneel. On the (in) security of the latest generation implantable cardiac defibrillators and how to secure them. In *Proceedings of the 32nd Annual Conference on Computer Security Applications*, pages 226–236. ACM, 2016.

[MV] Stephan Mueller and Marek Vasut. Linux kernel crypto api. https://www.kernel.org/doc/html/v4.12/crypto/index.html. Accessed: 2018-05-05.

[New] Lily Hay Newman. The ransomware meltdown experts warned about is here. https://www.wired.com/2017/05/ransomware-meltdown-experts-warned/. Accessed: 2018-03-15.

[oEU16] Council of European Union. Council regulation (eu) no 5419/2016, 2016. http://data.consilium.europa.eu/doc/document/ST-5419-2016-INIT/en/pdf.

[Pat90] Michael Quinn Patton. *Qualitative evaluation and research methods.* SAGE Publications, inc, 1990.

[PLN$^+$12] Helen Høgh Petersen, Mie Christa Jensen Larsen, Olav Wendelboe Nielsen, Finn Kensing, and Jesper Hastrup Svendsen. Patient satisfaction and suggestions for improvement of remote icd monitoring. *Journal of interventional cardiac electrophysiology*, 34(3):317–324, 2012.

[Pup] Lorenzo Pupillo. Software vulnerabilities disclosure: The european landscape. https://www.ceps.eu/publications/software-vulnerabilities-disclosure-european-landscape. Accessed 2018-05-21.

[RB17] Billy Rios and Jonathan Butts. Security evaluation of the implantable cardiac device ecosystem architecture and implementation interdependencies. 2017.

[res] Dimensional research. Trends in security framework adoption - a survey of it and security professionals. https://static.tenable.com/marketing/tenable-csf-report.pdf. Accessed: 2018-05-13.

[RW73] Horst W Rittel and Melvin M Webber. 2.3 planning problems are wicked. *Polity*, 4:155–169, 1973.

[San17] Karen Sandler. Cyborg lawyer 2.0, "hack proof", 2017. https://sfconservancy.org/blog/2017/apr/06/hack-proof/.

[Sch16] Suzzanne Schwartz. Postmarket management of cybersecurity in medical devices. 2016.

[Sho] FLOSS Shop. Openpgp smart card v3.3. https://www.floss-shop.de/en/security-privacy/smartcards/13/openpgp-smart-card-v3.3. Accessed: 2018-04-28.

[Sofa] Chilkat Software. C determine if a zip is encrypted or password-protected. https://www.example-code.com/csharp/zip_CheckForEncrypted.asp. Accessed: 2018-05-13.

[Sofb] Chilkat Software. Chilkat .net 9.4.0 download. http://download.informer.com/win-1192920767-4e5ef850-550846a3/chilkatdotnet4-9.4.0-win32.msi. Accessed 2018-03-12.

[Sofc] Chilkat Software. Ckzip c++ reference documentation. http://www.chilkatsoft.com/refdoc/vcCkZipRef.html#prop12. Accessed: 2018-05-13.

[Sofd] Chilkat Software. Creating chilkat activex components in delphi. https://www.chilkatsoft.com/p/p_153.asp. Accessed 2018-03-08.

[Sofe] Chilkat Software. Unzip a .zip archive. https://www.example-code.com/cpp/zip_SimpleUnzip.asp. Accessed: 2018-03-15.

[Soff] Kithara Software. Real-time for windows. http://kithara.com/en. Accessed: 2018-04-27.

[SOMM10] Karen Sandler, Lysandra Ohrstrom, Laura Moy, and Robert McVay. Killed by code: Software transparency in implantable medical devices. *Software Freedom Law Center*, pages 308–319, 2010.

[STW+07] Roy Small, Wilson Tang, William Wickemeyer, Robin Germany, Bobbi Hoppe, John Andriulli, Peter Brady, LaBeau Melody, and Douglas Hettrick. Managing heart failure patients with intra-thoracic impedance monitoring: a multi-center us evaluation. *Journal of Cardiac Failure*, 13(6):113–114, 2007.

[SVE+18] Leslie A. Saxon, Niraj Varma, Laurence M. Epstein, Leonard I. Ganz, and Andrew E. Epstein. Factors influencing the decision to proceed to firmware upgrades to implanted pacemakers for cybersecurity risk mitigation. *Circulation*, 2018.

[Tan] Ellen Tannam. Why cybersecurity is essential for the progress of medtech. Accessed 2018-05-13.

[Tre57] Marc-Adelard Tremblay. The key informant technique: A nonethnographic application. *American Anthropologist*, 59(4):688–701, 1957.

[VAMPR04] R Hevner Von Alan, Salvatore T March, Jinsoo Park, and Sudha Ram. Design science in information systems research. *MIS quarterly*, 28(1):75–105, 2004.

[Wie14] Roel J. Wieringa. *Design Science Methodology*. Springer, 2014.

[Win] Winitor. pestudio. https://www.winitor.com/. Accessed: 2018-05-13.

[WSK17] Anders Been Wilhelmsen and Eivind Skjelmo Kristiansen. Security testing of wireless body area networks with implantable medical devices, 2017.

[Yub] Yubico. Security key by yubico. https://www.yubico.com/product/security-key-by-yubico/. Accessed: 2018-04-28.

Appendix A

List Files Code

A.1 Edited For Readability

This code has been altered for readability. Working versions of the code has been given to the security project initiated by Dr. Moe. The order of functions has been altered, functions has been simplified to highlight their purpose, and graphics and print statements as been deleted.

A.1.1 List Files of Windows XP

The main script `clean_xp_tabulator.py` in listing A.1 imports the script `storage_parser.py` in listing A.2 and `tabulator.py` in listing A.3

```
import storage_parser
import tabulator
import os

output_file_name = 'CleanXPFileTable.tex'
output_file_path = '../sections/'
storage_folder = os.path.abspath('c:/')

# Parse storage for table content
storage_parser.parse_storage_folder_for_file_hashes_and_file_paths(
    storage_folder
)
storage_parser.parse_latex_file(
    output_file_path,
    output_file_name
)

# Create table
tabulator.categorize_hashes(
    storage_parser.hashes_from_storage,
    storage_parser.hashes_from_latex_file
)
tabulator.build_updated_table_content()
tabulator.write_updated_table_content_to_file(
    output_file_path,
    output_file_name
)
```

Listing A.1: clean_xp_tabulator.py

```
import subprocess
import os
import string

storage_folder = None
hashes_from_storage = {} # key=hash and value=path
hashes_from_latex_file = {} #key=hash and value=path
lines_from_latex_file = {} #key=hash and value=list[path,description]
duplicateFiles = set()

def parse_storage_folder_for_file_hashes_and_file_paths(
new_storage_folder):
    global storage_folder
    storage_folder = new_storage_folder
    # Get file paths
    file_paths = get_file_paths_in(storage_folder)
    # Get hashes of files
    generate_hashes_from_file_paths(hashes_from_storage, file_paths)

def get_file_paths_in(folder):
    # Setup function variables
    file_paths = set()
    storage_folder_path =
        os.path.join(
        os.path.dirname(os.path.realpath(__file__))
        , '..', folder)

    # Traverse target folder for subfolders and files
    for root, dirs, files in os.walk(storage_folder_path):
        base_path = string.split(root, folder)[1]
        for file in files:
            if file == ".DS_Store" or file == "\.DS_Store":
                print("\r" + graphics.Bcolors.WARNING
                + "Ignoring .DS_store file at: "
                + base_path + "/" + file
                + graphics.Bcolors.ENDC)
            else:
                file_paths.add(base_path + "\\" + file)
    return file_paths

def generate_hashes_from_file_paths(
    storage_dictionary,
    path_set
):
    # Setup function variables
    results = []
```

```python
    subfolder = '"../'
    if ":" in storage_folder: # Handle absolute path
        subfolder = '"'

    # Generate hashes
    for path in path_set:
        tmp = subprocess.Popen(
            'md5sum ' + subfolder + storage_folder + path + '"',
            shell=True,
            stdout=subprocess.PIPE, stderr=subprocess.STDOUT
        )
        print_var = tmp.stdout.read()
        tmp = print_var.split(" *")
        hash_value = tmp[0].strip()

        hash_of_empty_file =
            "d41d8cd98f00b204e9800998ecf8427e"
        if hash_value == hash_of_empty_file:
           # Ignore empty files
        else:
            results.append((hash_value, path))
    # Add hashes to dictionary
    storage_dictionary.update(results)
    identify_duplicate_hashes(storage_dictionary, results)

def identify_duplicate_hashes(storage_dictionary, results):
    if len(storage_dictionary) != len(results):
        seen_twice = list_duplicates(results)
        for hash_value in seen_twice:
            all_matching = filter(
                lambda x: x[0] == hash_value[0],
                results
            )
            duplicateFiles.add(tuple(all_matching))

def list_duplicates(seq):
    seen = set()
    seen_add = seen.add
    seen_twice = set(
        line for line in seq if
            line[0] in seen or seen_add(line[0])
    )
    return list(seen_twice)

def parse_latex_file(output_file_path, output_file_name):
    try:
        table_file = open(
            output_file_path + output_file_name, 'r'
        )
        # Table in latex file starts on line 24
        for i in xrange(24):
            try:
                table_file.next()
            except StopIteration:
                # File is empty
                break
        for line in table_file:
            values = string.split(
                line.replace(
                    "\_",
                    "_"
                ),
                '&'
            )
            if len(values) == 1: # End of latex table
                table_file.close()
                break
            path = values[0].strip()
            filename = values[1].strip()
            description = values[2].strip()
            hash_value = values[3][:-3].strip()
            # Store hash, file path and description
            lines_from_latex_file[hash_value] =
                [path + "/" + filename, description]
            # Store hash and file path
            hashes_from_latex_file[hash_value] =
                path + "/" + filename
    except IOError:
        # Latex file does not appear to exist
```

Listing A.2: storage_parser.py

```
from tabulate import tabulate
import sys
import storage_parser
import sys
import codecs
sys.stdout = codecs.getwriter('utf8')(sys.stdout)

updated_table_content = []  # Content to be written
added_files = {}  # Hashvalue
unchanged_files = {}  # Hashvalue
moved_files = {}  # Hashvalue
deleted_files = {}  # Hashvalue
changed_files = {}  # key = path, value = [newhash, oldhash]
duplicate_files = set()

def categorize_hashes(current_dict, past_dict):
    global added_files
    global unchanged_files
    global moved_files
    global deleted_files
    global changed_files

    dict_differ = DictDiffer(current_dict, past_dict)
    unchanged_files = dict_differ.unchanged()
    added_files = dict_differ.added()
    moved_files = dict_differ.moved()
    deleted_files = dict_differ.deleted()

    deleted_files_tmp = deleted_files.copy()
    added_files_tmp = added_files.copy()

    # Handle changed files (same hash different path)
    for hash_value_added in added_files_tmp:
        for hash_value_deleted in deleted_files_tmp:
            if (current_dict[hash_value_added]
                == past_dict[hash_value_deleted]
            ):
                changed_files[current_dict[hash_value_added]]
                = [hash_value_deleted, hash_value_added]

                added_files.remove(hash_value_added)
                deleted_files.remove(hash_value_deleted)

class DictDiffer(object):
    # DictDiffer from https://github.com/hughdbrown/dictdiffer
    def __init__(self, currentDict, pastDict):
        self.currentDict, self.pastDict = currentDict, pastDict
        self.setCurrent, self.setPast = set(currentDict.keys()), set(
    pastDict.keys())
        self.intersect = self.setCurrent.intersection(self.setPast)

    def added(self):
        return self.setCurrent - self.intersect

    def deleted(self):
        return self.setPast - self.intersect

    def moved(self):
        return set(o for o in self.intersect if self.pastDict[o] !=
    self.currentDict[o])

    def unchanged(self):
        return set(o for o in self.intersect if self.pastDict[o] ==
    self.currentDict[o])

def build_updated_table_content():
    for hash_value in sorted(added_files):
        file_path = storage_parser
            .hashes_from_storage[hash_value]
        description = ""
        path_elements = file_path.rsplit("\\", 1)
        folder_path = path_elements[0]
        file_name = path_elements[1]
        check_for_illegal_character(
            "&", folder_path, file_name
        )
        updated_table_content.append([
            folder_path,
            file_name,
            description,
            hash_value
        ])

    for hash_value in sorted(unchanged_files):
        line = storage_parser
            .lines_from_latex_file[hash_value]
        file_path = line[0]
        description = line[1]
        path_elements = file_path.rsplit("\\", 1)
        folder_path = path_elements[0]
        file_name = path_elements[1]
        updated_table_content.append([
            folder_path,
            file_name,
            description,
            hash_value
        ])

    for hash_value in sorted(moved_files):
        file_path = storage_parser
            .hashes_from_storage[hash_value]
        old_file_path = storage_parser
            .hashes_from_latex_file[hash_value]
        path_elements = file_path.rsplit("\\", 1)
        folder_path = path_elements[0]
        file_name = path_elements[1]
        check_for_illegal_character(
            "&", folder_path, file_name
        )
        description = storage_parser
            .lines_from_latex_file[hash_value][1]
        updated_table_content.append([
            folder_path,
            file_name,
            description,
            hash_value
        ])

    for file_path, hashList in sorted(changed_files.items()):
        description = storage_parser
            .lines_from_latex_file[hashList[0]][1]
        path_elements = file_path.rsplit("\\", 1)
        folder_path = path_elements[0]
        file_name = path_elements[1]
        updated_table_content.append([
            folder_path,
            file_name,
            description,
            hashList[1]
        ])

def check_for_illegal_character(
    illegal_character,
    folder_path,
    file_name
):
    if illegal_character in folder_path:
        sys.exit()
    if illegal_character in file_name:
        sys.exit()

def write_updated_table_content_to_file(
    output_filepath,
    output_filename
):
    output_file = open(
        output_filepath
        + output_filename,
        'w'
    )
    # Write everything needed before table content
    output_file.write(
        '%!TEX root = ../main.tex\n'
    )
    output_file.write(
        '% Please only edit descriptions\n'
    )
    output_file.write(
        '\\begin{landscape}\n\centering\n
        \\tabulinesep=_3pt^3pt\n
        \\begin{longtabu} to
        \linewidth{|X[1,l]|X[2,l]|X[1,l]H|}\n
        \caption{File List}\n\label{tab:file_list}\\\\\n
        \\tabucline[1pt]{1-}\n'
    )
    output_file.write(
        '\n\\textbf{Path} & \\textbf{Filename} &
        \\textbf{Description} & \\textbf{Hash} \\\\\n
        \\tabucline[0.5pt]{1-}\n
        \endfirsthead\n\caption{
        \\tablename --
        \\textit{Continued from previous page}} \\\\'
    )
    output_file.write(
        '\n\\tabucline[0.5pt]{1-}\n
        \\rowfont [c]{\\bfseries}\n
        Path & Filename & Description & Hash \\\\\n
        \\tabucline[0.5pt]{1-}\n\endhead\n\
        \tabucline[0.5pt]{1-}
        \multicolumn{4}{l}{\\textit{Continued on next page}}
        \\\\\endfoot\n
        \\tabucline[1pt]{1-}\n
        \endlastfoot\n
        \\taburowcolors[1]2{lightgrey..white}'
    )

    # Write table content
    output_file.write(
        tabulate(sorted(updated_table_content),
        tablefmt="latex").split('\hline', 2)[1]
    )
    # Write everything needed after table content
    output_file.write('\end{longtabu}\n\end{landscape}')
    output_file.close()
```

Listing A.3: tabulator.py

A.1.2 List Files and Signatures

The only differences between `main.py` in listing A.4 and `clean_xp_tabulator.py` from listing A.1 are the storage_parser and tabulator imported, which storage folder to parse and the output filename. The script can be used with different storage_folder and output_filename variables for the LaTeX table to accomodate parsing both the programmers disk partitions. It is also used for the table of interesting files in the `Bio\Execute\Tools` folder.

```
import signtool_parser as storage_parser
import signtool_tabulator as tabulator

output_file_name = 'DDiskFileTableWithSignatures.tex'
output_file_path = '..\\sections\\'
storage_folder = 'G:\\'

# Parse storage for table content
storage_parser.parse_storage_folder_for_file_hashes_and_file_paths(
    storage_folder
)
storage_parser.parse_latex_file(output_file_path, output_file_name)

# Create table
tabulator.categorize_hashes(
    storage_parser.hashes_from_storage,
    storage_parser.hashes_from_latex_file
)
tabulator.build_updated_table_content()
tabulator.write_updated_table_content_to_file(
    output_file_path,
    output_file_name
)
```

Listing A.4: main.py configured for the programmers D partition

The resulting LaTeX table from `clean_xp_tabulator.py` from listing A.1 is used by `signtool_parser.py` in listing A.5 to ignore file paths and names or hash values that has previously been parsed in a clean Windows XP installation.

```
import subprocess
import os
import string
import signtool_tabulator as tabulator

storage_folder = None
xp_latex_file_path = '../sections/CleanXPFileTable.tex'
hashes_from_xp_latex_file = set()
hashes_from_storage = {}
hashes_from_latex_file = {}
lines_from_latex_file = {}
hashes_with_signatures = {}
duplicateFiles = set()

def parse_storage_folder_for_file_hashes_and_file_paths(
    new_storage_folder
):
    global storage_folder
    global hashes_from_xp_latex_file
    storage_folder = new_storage_folder
    file_paths = get_file_paths_in(storage_folder)
    default_xp_file_paths = get_file_paths_from_latex_file(
        xp_latex_file_path
    )
    interesting_file_paths = file_paths.difference(
        default_xp_file_paths
    )
    hashes_from_xp_latex_file = get_hashes_from_latex_file(
        xp_latex_file_path
    )
    generate_hashes_from_file_paths(
        hashes_from_storage,
        interesting_file_paths
    )

def get_file_paths_in(folder):
    # Setup function variables
    file_paths = set()
    storage_folder_path =
        os.path.join(
            os.path.dirname(
                os.path.realpath(__file__)
            ),
            '..',
            folder
        )

    # Traverse target folder for subfolders and files
    for root, dirs, files in os.walk(storage_folder_path):
        base_path = string.split(root, folder)[1]
        for file in files:
            if file == ".DS_Store" or file == "/.DS_Store":
                # Ignore .DS_Store file
            else:
                file_paths.add((base_path + "\\" + file))
    return file_paths

def get_file_paths_from_latex_file(xp_latex_file_path):
    paths_from_xp_latex_file = set()
    try:
        table_file = open(xp_latex_file_path, 'r')
        for i in xrange(24): # Table starts at line 24
            try:
                table_file.next()
            except StopIteration:
                # Table is empty
                break
        for line in table_file:
            line = line.replace("textbackslash{}","")
            values = string.split(line.replace("\_", "_"), '&')
            if len(values) == 1: # End of table
                table_file.close()
                break
            path = values[0].strip()
            filename = values[1].strip()
            paths_from_xp_latex_file.add(path+"\\"+filename)
    except IOError:
        # File does not exist
    return paths_from_xp_latex_file

def get_hashes_from_latex_file(xp_latex_file_path):
    hashes_from_xp_latex_file = set()
    try:
        table_file = open(xp_latex_file_path, 'r')
        for i in xrange(24): # Table starts at line 24
            try:
                table_file.next()
            except StopIteration:
                # Table is empty
                break
        for line in table_file:
            line = line.replace("textbackslash{}","")
            values = string.split(line.replace("\_", "_"), '&')
            if len(values) == 1: # End of table
                table_file.close()
                break
            hash_value = values[3].strip()
            hashes_from_xp_latex_file.add(hash_value)
    except IOError:
        # File does not exist
    return hashes_from_xp_latex_file

def generate_hashes_from_file_paths(
    storage_dictionary,
    path_set
):
    # Setup function variables
    results = []
    signatures = []
    for path in path_set:
        # Generate hash
        md5 = subprocess.Popen(
            'md5sum "' + storage_folder + path + '"',
            shell=True,
            stdout=subprocess.PIPE
        )
        print_var = md5.stdout.read()
        md5 = print_var.split(" *")
        hash_value = md5[0].strip()[1:]
        hash_of_empty_file = "d41d8cd98f00b204e9800998ecf8427e"
        if hash_value == hash_of_empty_file:
            # Ignore empty files
            continue
        if hash_value in hashes_from_xp_latex_file:
            # Ignore original XP files
            continue
        else:
            results.append((hash_value, path))

        # Verify file signature
        signtool_verify = subprocess.Popen(
            "\"C:\\Program Files\\Microsoft Visual Studio\\2017\\
    Community\\Common7\\Tools\\VsDevCmd.bat\""
            # Need to add signtool.exe to path variable
            + ' && set PATH="C:\\Program Files\\Microsoft SDKs\\Windows
    \\v7.1\\Bin";%PATH%'
            + ' && signtool.exe verify /pa "'
            + storage_folder + path + '"',
            shell=True,
            stdout=subprocess.PIPE,
            stderr=subprocess.STDOUT
        )
        print_var = signtool_verify.stdout.read()

        # Filter signature verification result
        result = print_var.split("
    ***************************************")[2]
            .strip().split(": ")
        if len(result)==2:
            verify_result = result[0]
        if len(result)==3:
            verify_result = result[2]
        if len(result)==4:
            verify_result = result[2].strip() + ": " + result[3]
        verify_result = verify_result.replace("\r\n","").strip()
        verify_result = verify_result.replace("\r","").strip()
        verify_result = verify_result.replace(
            "This file format cannot be verified because it is not
    recognized.",
            "Unknown file format"
        )

        signatures.append((hash_value,verify_result))
    # Add hashes to dictionary
    storage_dictionary.update(results)
    identify_duplicate_hashes(storage_dictionary, results)
    # Add result from signature verification to dictionary
    hashes_with_signatures.update(signatures)

def list_duplicates(seq):
    seen = set()
    seen_add = seen.add
    seen_twice = set(line for line in seq if line[0] in seen or
    seen_add(line[0]))
    return list(seen_twice)


def identify_duplicate_hashes(storage_dictionary, results):
    if len(storage_dictionary) != len(results):
        seen_twice = list_duplicates(results)
        for hash_value in seen_twice:
            all_matching = filter(lambda x: x[0] == hash_value[0],
    results)
            duplicateFiles.add(tuple(all_matching))

def parse_latex_file(output_file_path, output_file_name):
    try:
        table_file = open(output_file_path + output_file_name, 'r')
        # Table in latex file starts on line 24
        for i in xrange(24):
            try:
                table_file.next()
            except StopIteration:
                # File is empty
                break
        for line in table_file:
            values = string.split(
                line.replace(
                    "\_",
                    "_"
                ),
                '&'
            )
            if len(values) == 1: # End of latex table
                table_file.close()
                break
            path = values[0].strip()
            filename = values[1].strip()
            description = values[2].strip()
            hash_value = values[3].strip()
            # Store has, file path and description
            lines_from_latex_file[hash_value] =
                [path + "\\" + filename, description]
            # Store has hand file path
            hashes_from_latex_file[hash_value] =
                path + "\\" + filename
    except IOError:
        # Latex file does not appear to exist
```

Listing A.5: signtool_parser.py

```
from tabulate import tabulate
import sys
import signtool_parser as storage_parser

updated_table_content = []  # Content to be written
added_files = {}  # Hashvalue
unchanged_files = {}  # Hashvalue
moved_files = {}  # Hashvalue
deleted_files = {}  # Hashvalue
changed_files = {}  # key = path, value = [newhash, oldhash]
duplicate_files = set()

def categorize_hashes(current_dict, past_dict):
    global added_files
    global unchanged_files
    global moved_files
    global deleted_files
    global changed_files

    dict_differ = DictDiffer(current_dict, past_dict)

    unchanged_files = dict_differ.unchanged()
    added_files = dict_differ.added()
    moved_files = dict_differ.moved()
    deleted_files = dict_differ.deleted()

    deleted_files_tmp = deleted_files.copy()
    added_files_tmp = added_files.copy()

    # Handle changed files (same hash different path)
    for hash_value_added in added_files_tmp:
        for hash_value_deleted in deleted_files_tmp:
            if current_dict[hash_value_added] == past_dict[
    hash_value_deleted]:
                changed_files[current_dict[hash_value_added]] = [
    hash_value_deleted, hash_value_added]
                added_files.remove(hash_value_added)
                deleted_files.remove(hash_value_deleted)

class DictDiffer(object):
    # DictDiffer from https://github.com/hughdbrown/dictdiffer

    def __init__(self, currentDict, pastDict):
        self.currentDict, self.pastDict = currentDict, pastDict
        self.setCurrent, self.setPast = set(currentDict.keys()), set(
    pastDict.keys())
        self.intersect = self.setCurrent.intersection(self.setPast)

    def added(self):
        return self.setCurrent - self.intersect

    def deleted(self):
        return self.setPast - self.intersect

    def moved(self):
        return set(o for o in self.intersect if self.pastDict[o] !=
    self.currentDict[o])

    def unchanged(self):
        return set(o for o in self.intersect if self.pastDict[o] ==
    self.currentDict[o])

def build_updated_table_content():
    for hash_value, path in (
        storage_parser.hashes_from_storage.items()
    ):
        if "\%" in path:
            storage_parser
                .hashes_from_storage[hash_value] =
                    path.replace("\%", "\textbackslash\%")
    replace =
        ('\\', '\\textbackslash{}'),
        ('_', '\_'),
        ('$', '\$'),
        (' ', '\ ')
    for hash_value in sorted(added_files):
        file_path = storage_parser
            .hashes_from_storage[hash_value]
        description = ""
        path_elements = file_path.rsplit("\\", 1)
        folder_path = path_elements[0]
        file_name = path_elements[1]
        check_for_illegal_character(
            "&", folder_path, file_name
        )
        folder_path = reduce(
            lambda a,
            kv:a.replace(*kv),
            replace,
            folder_path
        )
        file_name = reduce(
            lambda a,
            kv:a.replace(*kv),
            replace,
            file_name
        )
        updated_table_content.append([
            "\seqsplit{"+folder_path+"}",
            "\seqsplit{"+file_name+"}",
            description,
            hash_value,
            storage_parser.hashes_with_signatures[hash_value]
        ])

    for hash_value in sorted(unchanged_files):
        line = storage_parser
            .lines_from_latex_file[hash_value]
        file_path = line[0]
        description = line[1]
        path_elements = file_path.rsplit("\\", 1)
        folder_path = path_elements[0]
        file_name = path_elements[1]
        folder_path = reduce(
            lambda a,
            kv:a.replace(*kv),
            replace,
            folder_path
        )
        file_name = reduce(
            lambda a,
            kv:a.replace(*kv),
            replace,
            file_name
        )
        updated_table_content.append([
            "\seqsplit{"+folder_path+"}",
            "\seqsplit{"+file_name+"}",
            description,
            hash_value,
            storage_parser.hashes_with_signatures[hash_value]
        ])

    for hash_value in sorted(moved_files):
        file_path = storage_parser
            .hashes_from_storage[hash_value]
        old_file_path = storage_parser
            .hashes_from_latex_file[hash_value]
        path_elements = file_path.rsplit("\\", 1)
        folder_path = path_elements[0]
        file_name = path_elements[1]
        check_for_illegal_character(
            "&", folder_path, file_name
        )
        description = storage_parser
            .lines_from_latex_file[hash_value][1]
        folder_path = reduce(
            lambda a,
            kv:a.replace(*kv),
            replace,
            folder_path
        )
        file_name = reduce(
            lambda a,
            kv:a.replace(*kv),
            replace,
            file_name
        )
        updated_table_content.append([
            "\seqsplit{"+folder_path+"}",
            "\seqsplit{"+file_name+"}",
            description,
            hash_value,
            storage_parser.hashes_with_signatures[hash_value]
        ])

    for file_path, hashList in sorted(changed_files.items()):
        description = storage_parser
            .lines_from_latex_file[hashList[0]][1]
        path_elements = file_path.rsplit("\\", 1)
        folder_path = path_elements[0]
        file_name = path_elements[1]
        folder_path = reduce(
            lambda a,
            kv:a.replace(*kv),
            replace,
            folder_path
        )
        file_name = reduce(
            lambda a,
            kv:a.replace(*kv),
            replace,
            file_name
        )
        updated_table_content.append([
            "\seqsplit{"+folder_path+"}",
            "\seqsplit{"+file_name+"}",
            description,
            hashList[1],
            storage_parser
                .hashes_with_signatures[hashList[1]]
        ])

def check_for_illegal_character(
    illegal_character,
    folder_path,
    file_name
):
    if illegal_character in folder_path:
        sys.exit()
    if illegal_character in file_name:
        sys.exit()

def write_updated_table_content_to_file(
    output_filepath,
    output_filename
):
    output_file = open(
        output_filepath
        + output_filename,
        'w'
    )
    # Write everything needed before table content
    output_file.write('%!TEX root = ../main.tex\n')
    output_file.write('% Please only edit descriptions\n')
    output_file.write(
        '\\begin{landscape}\n\centering\n
        \\tabulinesep=_3pt^3pt\n
        \\begin{longtabu}
        to \linewidth{|X[1.5,l]|X[1.5,l]|X[1,l]H|X[1,l]|}\n
        \caption{File List}\n
        \label{tab:file_list}\\\\\n
        \\tabucline[1pt]{1-}\n'
    )
    output_file.write(
        '\n\\textbf{Path} & \\textbf{Filename} &
        \\textbf{Description} & \\textbf{Hash} &
        \\textbf{Signtool verification} \\\\\n
        \\tabucline[0.5pt]{1-}\n
        \endfirsthead\n
        \caption{\\tablename --
        \\textit{Continued from previous page}} \\\\'
    )
    output_file.write(
        '\n\\tabucline[0.5pt]{1-}\n
        \\rowfont [c]{\\bfseries}\n
        Path & Filename & Description &
        Hash & Signtool Verification \\\\\n
        \\tabucline[0.5pt]{1-}\n
        \endhead\n
        \\tabucline[0.5pt]{1-}
        \multicolumn{5}{l}{\\textit{Continued on next page}}
        \\\\\endfoot\n
        \\tabucline[1pt]{1-}\n
        \endlastfoot\n
        \\taburowcolors[1]2{lightgrey..white}'
    )
    # Write table content
    output_file.write(
        tabulate(
            sorted(updated_table_content),
            tablefmt="latex_raw"
        ).split('\hline', 2)[1]
    )
    # Write everything needed after table content
    output_file.write(
        '\end{longtabu}\n
        \end{landscape}'
    )
    output_file.close()
```

Listing A.6: signtool_tabulator.py

```
#
###### Requirements for macOS
#
#brew install md5sha1sum

#
###### Requirements without version specifiers ######
#
tabulate
progress
tqdm
```

Listing A.7: requirements.txt

A.2 Performance

Utility was prioritized above performance since the script itself is outside the scope of this thesis. However, since the script takes 4-5 hours to complete it is reasonable to address that the performance can be improved. One suggestion is replacing MD5 as the hash algorithm with for instance xxHash, reported to be 16 times faster https://github.com/Cyan4973/xxHash(sourced: 27. april).

Signature verification with Signtool takes about 1.4 seconds per file, and only files that are not part of a default Windows XP installation is verified.

Appendix B Description of Drivers and Functions in Kithara Base Driver

The following are translated descriptions of the drivers that apperad on the Kithara web page in 2002.

About the **base driver**: *'Basic package, always required, functions to open the driver, to find error descriptions, version control, debug aids'*

*'The **I/O Accelerator** is a tool for direct access to I/O ports and physical memory as well as the identification of PCI data and interface resources. The accesses are made quickly and directly to any I/O ports.'*

*'The **»Hardware Toolkit«** expands the »I/O Accelerator« with powerful mechanisms for interrupt programming. This provides all the necessary mechanisms for the rapid development of hardware drivers.'*

*'The **Timer Toolkit** is a tool for creating time-critical applications and real-time controls. It provides accurate timer routines and functions for high-resolution time measurements.'*

Description of functions found after reverse engineering the Kithara Base Driver with Ida Pro are sourced from the Kithara documentation of the Base and Kernel Driver at http://kithara.com/en/docs/krts:modules:base and http://kithara.com/en/docs/krts:modules:kernel. One function missing description is the function `KS_registerRtxAddress`.,

KS_startKernel	*Starts the kernel driver*
KS_stopKernel	*Stops the kernel driver*
KS_resetKernel	*Reset the kernel during development*
KS_getDriverVersion	*Determine the driver version*
KS_openDriver	*Open the driver*

KS_closeDriver	*Close the driver*
KS_getThreadPrio	*Gives the absolute priority of the current thread*
KS_setThreadPrio	*Sets the absolute priority of the current thread*
KS_createThread	*Creates a Windows application thread*
KS_createThreadEx	*Creates a Windows application thread*
KS_removeThread	*Exits from a Windows application thread*
KS_createSharedMem	*Creates shared memory for data exchange*
KS_freeSharedMem	*Frees shared memory*
KS_getSharedMem	*Gives the current address of shared memory*
KS_prepareKernelExec	*prepares user-space code for kernel-space execution*
KS_endKernelExec	*Frees resources formerly allocated by KS_prepareKernelExec*
KS_testKernelExec	*Runs code on kernel-level*
KS_registerRtxAddress	
KS_execSyncFunction	*Executes a function synchronized with an interrupt handler*
KS_createEvent	*Creates an event object. To block Windows threads or tasks.*
KS_closeEvent	*Removes an event object.*
KS_setEvent	*Sets the event*
KS_resetEvent	*Resets the event*
KS_pulseEvent	*Sets and immediately resets the event*
KS_waitForEvent	*Blocks the current thread or task for an event*
KS_getEventState	*Gives the status of an event*
KS_postMessage	*Posts a message to a window*
KS_getErrorStringEx	*Error code in plain text*
KS_getErrorString	*Error code in plain text*
KS_createKernelCallBack	*Creates a callback object on a kernel-DLL function*
KS_createCallBackEx	*Creates a callback object for execution on application level or kernel level*
KS_createCallBack	*Creates a callback object for execution on application level or kernel level*
KS_removeCallBack	*Removes a callback*
KS_execCallBack	*Executes a callback*
KS_getCallState	*Gives information about a signal handler*

KS_enterQuietSection	*Starts execution of ring-3 (application level) code on ring-0 (kernellevel)*
KS_releaseQuietSection	*Ends execution of ring-3 code on ring-0*
KS_loadKernel	*Loads a DLL in a specific address-space*
KS_freeKernel	*Unloads a DLL from memory*
KS_execKernelFunction	*Executes a function of a kernel-DLL by name*
KS_logMessage	*Generates a log message in the Kernel Tracer*
KS_makeBeep	*Beeps shortly over the speaker*

Table B.1: Descriptions of functions in Kithara

Appendix C
List of PDB Paths

```
c:\EgsV01\System\PMSpecificComponents\Belos\HolterDetailsBelos\Release\
    HolterDetailsBelos.pdb
c:\EgsV01\System\PMSpecificComponents\Belos\HolterViewBelos\Release\
    HolterViewBelos.pdb
c:\EgsV01\System\PMSpecificComponents\Primus\AppPrimusParmTransmit\
    Release\AppPrimusParmTransmit.pdb
c:\EgsV01\System\PMSpecificComponents\Primus\AppPrimus\Release\
    AppPrimus.pdb
c:\EgsV01\System\PMSpecificComponents\Primus\BioFUPHistory\Release\
    BioFUPHistory.pdb
c:\EgsV01\System\PMSpecificComponents\Primus\CommunicationPrimus\
    Release\CommunicationPrimus.pdb
c:\EgsV01\System\PMSpecificComponents\Primus\CurrentCalcPrimus\Release\
    CurrentCalcPrimus.pdb
c:\EgsV01\System\PMSpecificComponents\Primus\HolterPrimus\Release\
    HolterPrimus.pdb
c:\EgsV01\System\PMSpecificComponents\Primus\IdentificationPrimus\
    Release\IdentificationPrimus.pdb
c:\EgsV01\System\PMSpecificComponents\Primus\IegmPrimus\Release\
    IegmPrimus.pdb
c:\EgsV01\System\PMSpecificComponents\Primus\NavigationPrimus\Release\
    NavigationPrimus.pdb
c:\EgsV01\System\PMSpecificComponents\Primus\OptionsPrimus\Release\
    OptionsPrimus.pdb
c:\EgsV01\System\PMSpecificComponents\Primus\ParametersLayoutPrimus\
    Release\ParametersLayoutPrimus.pdb
c:\EgsV01\System\PMSpecificComponents\Primus\PrimusCore\Release\
    PrimusCore.pdb
c:\EgsV01\System\PMSpecificComponents\Primus\PrimusRules\R1\Release\
    PrimusRulesR1.pdb
c:\EgsV01\System\PMSpecificComponents\Primus\StatisticsPrimus\Release\
    StatisticsPrimus.pdb
C:\EgsV01\System\PMSpecificComponents\Stratos\BioAppStratos\
    ReleaseUMinDependency\BioAppStratos.pdb
C:\EgsV01\System\PMSpecificComponents\Stratos\BioEriCalculation\
    ReleaseUMinSize\BioEriCalculation.pdb
```

```
C:\EgsV01\System\PMSpecificComponents\Stratos\BioStratosCommunication\
    ReleaseUMinSizeUSB\BioStratosCommunicationUSB.pdb
C:\EgsV01\System\PMSpecificComponents\Stratos\StratosCommonCom\
    ReleaseUMinSize\StratosCommonCom.pdb
C:\EgsV01\System\PMSpecificComponents\Stratos\StratosCommon\DebugU\
    StratosCommond.pdb
C:\EgsV01\System\PMSpecificComponents\Stratos\StratosCommon\
    ReleaseUMinSize\StratosCommon.pdb
C:\EgsV01\System\PMSpecificComponents\Stratos\StratosHolter\
    ReleaseUMinDependency\StratosHolter.pdb
C:\EgsV01\System\PMSpecificComponents\Stratos\StratosIdentification\
    ReleaseUMinSize\StratosIdentification.pdb
C:\EgsV01\System\PMSpecificComponents\Stratos\StratosIegm\
    ReleaseUMinSizeUSB\StratosIegmUSB.pdb
C:\EgsV01\System\PMSpecificComponents\Stratos\StratosOptions\
    ReleaseUMinDependency\StratosOptions.pdb
C:\EgsV01\System\PMSpecificComponents\Stratos\StratosParameters\
    ReleaseUMinDependency\StratosParameters.pdb
C:\EgsV01\System\PMSpecificComponents\Stratos\StratosRules\2B\
    ReleaseUMinSize\StratosRules2B.pdb
C:\EgsV01\System\PMSpecificComponents\Stratos\StratosRules\2C\
    ReleaseUMinSize\StratosRules2C.pdb
C:\EgsV01\System\PMSpecificComponents\Stratos\StratosStatistics\
    ReleaseUMinSize\StratosStatistics.pdb
c:\EgsV01\System\PMSpecificComponents\Tach35\AppTach35\Release\
    AppTach35.pdb
c:\EgsV01\System\PMSpecificComponents\Tach35\CommunicationTach35\
    Release\CommunicationTach35.pdb
c:\EgsV01\System\PMSpecificComponents\Tach35\HolterTach35\Release\
    HolterTach35.pdb
c:\EgsV01\System\PMSpecificComponents\Tach35\IdentificationTach35\
    Release\IdentificationTach35.pdb
c:\EgsV01\System\PMSpecificComponents\Tach35\IegmTach35\Release\
    IegmTach35.pdb
c:\EgsV01\System\PMSpecificComponents\Tach35\NavigationTach35\Release\
    NavigationTach35.pdb
c:\EgsV01\System\PMSpecificComponents\Tach35\OptionsTach35\Release\
    OptionsTach35.pdb
c:\EgsV01\System\PMSpecificComponents\Tach35\ParametersTach35\Release\
    ParametersTach35.pdb
c:\EgsV01\System\PMSpecificComponents\Tach35plus\AppTach35plus\Release\
    AppTach35plus.pdb
c:\EgsV01\System\PMSpecificComponents\Tach35plus\
    IdentificationTach35plus\Release\IdentificationTach35plus.pdb
c:\EgsV01\System\PMSpecificComponents\Tach35plus\ParametersTach35plus\
    Release\ParametersTach35plus.pdb
c:\EgsV01\System\PMSpecificComponents\Tach35plus\StatisticsTach35plus\
    Release\StatisticsTach35plus.pdb
c:\EgsV01\System\PMSpecificComponents\Tach35plus\Tach35plusRules\1G\
    Release\Tach35plusRules1G.pdb
c:\EgsV01\System\PMSpecificComponents\Tach35\StatisticsTach35\Release\
    StatisticsTach35.pdb
```

```
c:\EgsV01\System\PMSpecificComponents\Tach35\Tach35Rules\1G\Release\
    Tach35Rules1G.pdb
c:\EgsV01\System\SharedComponents\BioChannelView\Release\BioChannelView
    .pdb
c:\EgsV01\System\SharedComponents\BioNLS\Release\BioNLS.pdb
c:\EgsV01\System\SharedComponents\BioParam\Release\BioParam.pdb
c:\EgsV01\System\SharedComponents\BioPrintServer\Release\BioPrintServer
    .pdb
c:\EgsV01\System\SharedComponents\BioScriptHelper\Release\
    BioScriptHelper.pdb
c:\EgsV01\System\SharedComponents\BioSignalMgr\Release\BioSignalMgr.pdb
c:\EgsV01\System\SharedComponents\BioStatistics\Release\BioStatistics.
    pdb
c:\EgsV01\System\SharedComponents\BioStudyManagement\Release\
    BioStudyManagement.pdb
c:\EgsV01\System\SharedComponents\ECGMark\Release\ECGMark.pdb
c:\EgsV01\System\SharedComponents\ECGSignal\Release\ECGSignal.pdb
c:\EgsV01\System\SharedComponents\ErrorHandlerHelper\Release\
    ErrorHandlerHelper.pdb
c:\EgsV01\System\SharedComponents\ParameterHelper\Release\
    ParameterHelper.pdb
c:\EgsV01\System\SharedComponents\PrintHelper\Release\PrintHelper.pdb
c:\EgsV01\System\SharedComponents\RealTimeDataProcessing\RtdsPushData\
    Release\RtdsPushData.pdb
c:\EgsV01\System\SharedUIControls\BioCounterUI\Release\BioCounterUI.pdb
c:\EgsV01\System\SharedUIControls\BioDataExplorerUI\Release\
    BioDataExplorer.pdb
c:\EgsV01\System\SharedUIControls\BioDynAVEx\Release\BioDynAVEx.pdb
c:\EgsV01\System\SharedUIControls\BioDynAV\Release\BioDynAV.pdb
c:\EgsV01\System\SharedUIControls\BioECGControls\Release\BioECGControls
    .pdb
c:\EgsV01\System\SharedUIControls\BioEdit\Release\BioEdit.pdb
c:\EgsV01\System\SharedUIControls\BioFlash\Release\BioFlash.pdb
c:\EgsV01\System\SharedUIControls\BioGauge\Release\BioGauge.pdb
c:\EgsV01\System\SharedUIControls\BioGrid\Release\BioGrid.pdb
c:\EgsV01\System\SharedUIControls\BioMessageUI\Release\BioMessageUI.pdb
c:\EgsV01\System\SharedUIControls\BioNavigator\Release\BioNavigator.pdb
c:\EgsV01\System\SharedUIControls\BioOfflineEcgViewer\Release\
    BioOfflineEcgViewer.pdb
c:\EgsV01\System\SharedUIControls\BioOnlineECG\Release\BioOnlineECG.pdb
c:\EgsV01\System\SharedUIControls\BioParameterUI\Release\BioParameterUI
    .pdb
c:\EgsV01\System\SharedUIControls\BioPDFView\Release\BioPDFView.pdb
c:\EgsV01\System\SharedUIControls\BioProgramSet\Release\BioProgramSet.
    pdb
c:\EgsV01\System\SharedUIControls\BioReleaseCodeUI\Release\
    BioReleaseCodeUI.pdb
c:\EgsV01\System\SharedUIControls\BioStudyManagementUI\Release\
    BioStudyManagementUI.pdb
c:\EgsV01\System\SharedUIControls\BioTableUI\Release\BioTableUI.pdb
c:\EgsV01\System\SharedUIControls\BioTableView\Release\BioTableView.pdb
c:\EgsV01\System\SharedUIControls\BioTrendUI\Release\BioTrendUI.pdb
```

```
c:\EgsV01\System\SharedUIControls\BioTrendViewUI\Release\BioTrendViewUI
    .pdb
c:\EgsV01\System\SharedUIControls\ContextSensitiveHelp\Release\
    ContextSensitiveHelp.pdb
c:\EgsV01\System\SharedUIControls\HolterUI\FreezeDialog\Release\
    FreezeDialog.pdb
c:\EgsV01\System\SharedUIControls\HolterUI\HolterModel\Release\
    HolterModel.pdb
c:\EgsV01\System\SharedUIControls\HolterUI\HolterPrint\Release\
    HolterPrint.pdb
c:\EgsV01\System\SharedUIControls\HolterUI\IntervalView\Release\
    IntervalView.pdb
c:\EgsV01\System\SharedUIControls\HolterUI\RealTimeDataView\Release\
    RealTimeDataView.pdb
c:\EgsV01\System\SharedUIControls\ParamsetConflictUI\Release\
    ParamsetConflictUI.pdb
c:\EgsV01\System\SharedUIControls\ProgressEXE\Release\ProgressEXE.pdb
c:\EgsV01\System\SharedUIControls\TabUI\Release\TabUI.pdb
c:\EgsV01\System\StartupSystem\EGSStart\Release\EGSStart.pdb
C:\EgsV01\Tools\DIS_TEST\DIS_Test.pdb
```

Appendix D

Chilkat ZIP Encryption Detection Script

```python
import sys
import chilkat
import os

zip = chilkat.CkZip()

#  Any string unlocks the component for the 1st 30-days.
success = zip.UnlockComponent("foo")
if (success != True):
    print(zip.lastErrorText())
    sys.exit()

#  An encrypted or password-protected zip can be opened
#  without specifying the password.  However, the contents
#  of the files cannot be unzipped without providing the correct
#  password.

#
success = zip.OpenZip("2018_02_25_07_18_44___402_0.zip")
if (success != True):
    print(zip.lastErrorText())
    sys.exit()

#  If the zip is password-protected, meaning that it uses
#  the old (insecure) Zip 2.0 encryption, then the
#  PasswordProtect property will be True
bPwdProt = zip.get_PasswordProtect()
if (bPwdProt):
 print("This zip is password-protected.")
else:
 print("This zip is NOT password-protected.")

#  If the zip is AES encrypted (WinZip compatible) then
#  the Encryption property will be equal to 4.
encValue = zip.get_Encryption()
if (encValue == 4):
 print("This zip is AES encrypted.")
```

```
elif (encValue == 0):
 print("This zip is not encrypted.")
else:
 print("The enc value is: "+encValue)

#  If the Encryption property = 0, then the zip is NOT
#  strong encrypted, and is either password-protected or
#  entirely unencrypted, depending on the value of the
#  PasswordProtect property.

#  If the Encryption property = 1, 2, or 3, then the zip was
#  encrypted using AES, Blowfish, or Twofish using a
#  Chilkat-specific encryption format that was implemented
#  prior to the publication of the Zip AES standard.

unzipCount = zip.Unzip('ChilkatUnzipped')
if (unzipCount < 0):
    print(zip.lastErrorText())
else:
    print("Success!")

print "Amount of files in zip:",unzipCount
zip.CloseZip()
```

Appendix E Chilkat C# ZIP Script

```
  using    System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Chilkat;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            //Chilkat.Crypt2 crypt2 = new Chilkat.Crypt2();
            Chilkat.Zip zip = new Chilkat.Zip();
            zip.UnlockComponent("HackThatPaceMaker");
            string zipFilename = "C:\\Documents and Settings\\
    Administrator\\Desktop\\2016_09_22_08_37_47___402_0.zip";
            zip.DecryptPassword = "BIOTRONIK";
            bool pwok = zip.VerifyPassword();
            if (pwok)
            {
                Console.WriteLine("DINGDINGDING!");
            }
            else
            {
                Console.WriteLine(zip.LastErrorText);
                return;
            }

            bool success = zip.OpenZip(zipFilename);
            if (!success)
            {
                Console.WriteLine(zip.LastErrorText);
                return;
            }
            bool isUnZipped = zip.Extract("C:\\Documents and Settings\\
    Administrator\\Desktop\\please");
            if (!isUnZipped)
            {
                Console.WriteLine(zip.LastErrorText);
                return;
            }
        }
    }
}
```

```
  using    System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Chilkat;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            Chilkat.Zip zip = new Chilkat.Zip();
            zip.UnlockComponent("HackThatPaceMaker");
            string zipFilename = "C:\\Documents and Settings\\
    Administrator\\Desktop\\test.zip";
            zip.Encryption = 3;
            zip.EncryptKeyLength = 256;
            zip.EncryptPassword = "BIOTRONIK";
            bool success;
            success = zip.NewZip(zipFilename);
            if (!success)
            {
                Console.WriteLine(zip.LastErrorText);
                return;
            }

            bool recurse = true;
            success = zip.AppendFiles("C:\\Documents and Settings\\
    Administrator\\Desktop\\please", recurse);
            if (!success)
            {
                Console.WriteLine(zip.LastErrorText);
                return;
            }
            success = zip.WriteZipAndClose();
            if (!success)
            {
                Console.WriteLine(zip.LastErrorText);
                return;
            }
            Console.WriteLine("Created test.zip");
        }
    }
}
```

Appendix F

JTAGulator UART Scan Data

```
JTAGulating! Press any key to abort.........................
TXD: 2
RXD: 3
Baud: 900
Data: . [ EF ]

TXD: 2
RXD: 3
Baud: 1200
Data: . [ FF ]

TXD: 2
RXD: 3
Baud: 1800
Data: . [ FF ]

TXD: 2
RXD: 3
Baud: 2400
Data: . [ BF ]

TXD: 2
RXD: 3
Baud: 3600
Data: . [ EF ]

TXD: 2
RXD: 3
Baud: 4800
Data: . [ FF ]

TXD: 2
RXD: 3
Baud: 7200
Data: . [ FF ]

TXD: 2
```

```
RXD: 3
Baud: 9600
Data: . [ FF ]

TXD: 2
RXD: 3
Baud: 14400
Data: . [ FF ]

TXD: 2
RXD: 3
Baud: 19200
Data: . [ FF ]

TXD: 2
RXD: 3
Baud: 28800
Data: . [ EF ]

TXD: 2
RXD: 3
Baud: 31250
Data: . [ FF ]

TXD: 2
RXD: 3
Baud: 38400
Data: . [ FF ]

TXD: 2
RXD: 3
Baud: 57600
Data: . [ FD ]

TXD: 2
RXD: 3
Baud: 76800
Data: } [ 7D ]

TXD: 2
RXD: 3
Baud: 115200
Data: . [ 7F ]

TXD: 2
RXD: 3
Baud: 153600
Data: . [ FF ]

TXD: 2
RXD: 3
Baud: 250000
```

```
Data: . [ FF ]

TXD: 2
RXD: 3
Baud: 307200
Data: . [ FF ]
...
TXD: 2
RXD: 6
Baud: 900
Data: . [ 1F ]

TXD: 2
RXD: 6
Baud: 1200
Data: - [ 2D ]

TXD: 2
RXD: 6
Baud: 1800
Data: - [ 2D ]

TXD: 2
RXD: 6
Baud: 2400
Data: . [ 0D ]

TXD: 2
RXD: 6
Baud: 3600
Data: O [ 4F ]

TXD: 2
RXD: 6
Baud: 4800
Data: . [ 0D ]

TXD: 2
RXD: 6
Baud: 7200
Data: M [ 4D ]

...snip...
```

Appendix G

Interview Guide

1. **Kan vi starte med at du beskriver hva du gjør på sykehuset?**

Oppfølgingsspørsmål:

– Hva er dine ansvarsområder?

Formål:

– *Gi en komfortabel, ikke-truende start på intervjuet*
– *Lokalisere personen i organisasjonen fra hans/hennes eget perspektiv*

2. **Kan du beskrive for meg hvilke andre roller som er involvert med pacemakere og programmerere her på sykehuset?**

Oppfølgingsspørsmål:

– Hva er de andre rollene sitt ansvar?

Formål:

– *Finner andre relevante roller i organisasjonen fra hans/hennes perspektiv*
– *Kandidaten differensierer sin rolle og spør hva kandidaten selv og andre er ansvarlig for.*

3. **Kan du ta meg gjennom hva som skjer etter at at man har bestemt seg for å operere inn en pacemaker?**

Oppfølgingsspørsmål:

– Hvilke roller er involvert?
– Hvordan velger man merke og modell av pacemakeren?

Formål:

– *Gi intervjueren mulighet til å grave i faktorer som kandidaten mener er relevant etter at man har bestemt seg for å operere inn en pacemaker.*

4. **Kan du ta meg gjennom pacemakeren sin reise fra leverandøren til den blir operert inn i kroppen?**

Formål:

– *Gi intervjueren muligheten til å utforske et bredt spekter av faktorer som kandidaten anser som relevant for en pacemaker under dens reise fra leverandør til pasient.*

Oppfølgingsspørsmål:

– Hvilke roller er involvert?
– Hvordan er pacemakeren lagret på sykehuset?
– Hvem har tilgang til pacemakeren på denne reisen?
– Blir pacemakeren testet på noen måte før operasjon?

Antagelser:

– Den kommer i en pose
– Noen bekrefter ID-en/serienummer og modell til pacemakeren
– Det følges en protokoll under operasjonen

5. **Kan du ta meg gjennom hvordan en pacemaker blir konfigurert?**

   Oppfølgingsspørsmål:

– Hvem tar seg av ansvaret for konfigurasjon av pacemakeren?
– Hvordan sjekkes det opp at alt er som det skal, hva er prosedyre for konfigurasjonsdata?

6. **Hva skjer når en pasient som har pacemaker blir lagt inn på sykehuset med hjerteproblemer? Kan du ta meg gjennom hva som skjer da?**

Formål:

– *Gi intervjueren muligheten til å utforske et bredt spekter av faktorer som kandidaten mener er relevant for en pasient med et hjerteproblem som blir lagt inn som også har pacemaker.*

Oppfølgingsspørsmål:

– Hvilke roller er involvert?
– Hvordan vet man hvilken programmerer man skal bruke?
– Hvordan finner du informasjon om pasienten?
– Hvordan finner du informasjon om pasienten ikke hører til ved sykehuset?
– Hvordan bruker dere programmereren?
– Hva ser dere etter i dataene?
– Kan dere se noen form for historie fra pacemakeren?
– Vet de hvor dataen kommer fra?

7. **Kan du ta meg gjennom en vanlig oppfølging av en pasient med pacemaker?**

Oppfølgingsspørsmål:

– Hva ser du etter når du følger opp en pasient med pacemaker?
– Kan du gi noen eksempler på hva som ville vært unormalt?

8. **Vi har forstått det slik at det er risiko involvert med ledningene til pacemakeren, og at 200 pasienter fikk ledninger fjernet i fjor. Kan du fortelle oss hvorfor ledninger blir fjernet?**

Oppfølgingsspørsmål:

- Kan du beskrive hvilken risiko pasienten utsettes for når man fjerne ledninger?
- På hvilken bakgrunn blir avgjørelsen om å fjerne ledninger tatt?

Formål:

- *Se om avgjørelsen om å operere kommer utelukkende fra programmereren*

9. **Hvis vi nå går over til programmereren. Kan du beskrive til meg hvordan programmereren blir brukt her på sykehuset?**

Oppfølgingsspørsmål:

- Står den på en fast plass?
- Hvilke roller er det som bruker programmereren?

10. **Kan du fortelle meg hvilken opplæring som kreves for å bruke en programmerer?**

Oppfølgingsspørsmål:

- Hvordan gjennomføres opplæringen?
- Gis det noen ekstra opplæring ift. ansvar/roller?

11. **Hvordan lagrer dere data fra programmereren?**

Oppfølgingsspørsmål:

- Hvordan blir data fra programmereren lagret i et journalsystem??

12. **Printer dere noen gang fra programmereren?**

Oppfølgingsspørsmål:

- Hvordan blir det utført?
- Hvordan blir dataene som blir printet håndtert?

13. **Hvis en pasient spør om å få sine data, har dere noen protokoll for det?**

Oppfølgingsspørsmål:

- Kan du beskrive en type data de kan få, og en type data de ikke kan få?
- Hvem er sykehusets databehandlere? Hvem behandler personopplysninger fra programmerer på vegne av sykehuset?

14. **Vet du omtrent hvor mange pasienter som ligger inn på** programmereren?

Oppfølgingsspørsmål:

- Når slettes pasientdata fra programmereren?
- Er det noen rutiner for sletting av data?

15. **Tar noen av programmererne i bruk Bluetooth på noen måte?**

Oppfølgingsspørsmål:

- Om de bruker bluetooth, til hva brukes det til?

16. **Er noen av programmererne koblet på internett?**

Oppfølgingsspørsmål:

- Er du klar over om de kan kobles på internett eller ei?

17. **Hva er rutinene for å slette data?**

18. **Hva skjer med en programmerer som ikke brukes lenger?**

Oppfølgingsspørsmål:

- Hva skjer rent fysisk med programmereren, slettes data f.eks?
- Tar leverandøren tilbake programmereren?
- Overføres data til en ny programmerer?
- Isåfall, hvordan gjøres dette, og av hvem?

19. **Kan du beskrive hvordan du beskytter programmereren mot uautorisert tilgang?**

Oppfølgingsspørsmål:

- Tas det noen sikkerhetshensyn ved oppstart av programmereren?
- Hvor er det pacemaker programmererne befinner seg på sykehuset?
- Er de noen gang ulåste eller lett tilgjengelige?
- Er det mulig for oss å bare gå opp til programmerer?

20. **Brukes det USB-sticks med programmereren?**

Oppfølgingsspørsmål:

- Hva brukes USB-stickene til?
- Hvor kommer USB-stickene fra?
- Brukes de i andre maskiner?
- Isåfall hvilke andre maskiner?
- Er det en dedikert USB-stick?
- Hvor befinner USB-sticken seg?
- Har USB-sticken merkelapp?
- Forlater USB-sticken noen gang sykehuset?
- Hvem har tilgang til USB-sticken?
- Hvem bruker USB-sticken?
- Er USB-sticken kryptert?

21. **Hva er konsekvensen hvis en programmerer på sykehuset ikke kan brukes?**

Oppfølgingsspørsmål:

- Er det noen tidspunkt hvor det er mer uheldig at programmereren ikke kan brukes?

22. **Kan du beskrive hvordan en HMU er koblet til og konfigurert med en programmerer?**

Oppfølgingsspørsmål:

- Hvordan velges modellen av en HMU?

23. **Kan du beskrive hvordan dataen fra en HMU brukes?**

24. **Har det blitt gjort tiltak for å forbedre sikkerheten rundt rutiner, prosedyrer eller utstyr som er en del av pacemaker økosystemet?**

25. **Hvordan kommer patching, eller oppdateringer for pacemakeren til programmereren?**

26. **Er det slik at noen noen pasienter ikke får oppdateringer? F.eks. Hvis oppdateringsprosessen i seg selv utgjør en risiko for pasienten?**