

SOFTWARE PROTOTYPE FOR PACEMAKER INTRUSION DETECTION SYSTEM

By

Euan Brook

A Project

Submitted to the School of Digital, Technologies and Arts

Staffordshire University

For the degree of BSc (Hons) Cyber Security

Supervised by Adam Jacobs

May 2021

CONTENTS

1. Introduction	5
2. Background	5
3. Functional Requirements	6
4. Project Management	7
5. Lit Review	8
5.1 Pacemakers	8
5.2 Attack Mitigation	9
5.3 Microprocessors	9
5.4 Intrusion System	12
5.5 Connectivity	12
5.6 Encryption	12
5.6.1 TwoFish	13
5.6.2 Diffie-Hellman	13
5.6.3 AES	13
5.6.4 DSA	13
5.7 Language	14
5.7.1 The Learning Curve	14
5.7.2 Low-level & High-level	14
5.7.3 Picking a language	15
6. Analysis	15
7. Artefact Development	16
7.1 Design	16
7.2 Development & Implementation	16
7.2.1 The Base Pacemaker	17
7.2.2 The Logging System	20
7.2.3 System Admin	21
7.2.4 Connectivity	23
7.3 Prototype Pacemaker	24
7.3.1 main.py	25
7.3.2 SysAdmin.py	25
7.3.3 Logger.py	25

7.3.4 ppmServer.py	25
7.3.5 ppmClient.py	25
8. Artefact testing/Eval	26
8.1 Testing	26
8.2 Evaluation	28
9. Conclusion.....	29
References	30
Appendix.....	32

Abstract - This Paper will be investigating, reviewing and designing pacemaker systems and reviewing previous solutions to implement an intrusion detection system into a prototype pacemaker that will be capable of ensuring device and patient security. The investigation will be able to provide a further understanding as to what tools should be used to develop an intrusion detection system while adhering to device limitations such as battery size and processing power. The paper will also review upcoming technology that can assist with battery size constraints. ~

~The results found from the investigation into pacemakers and security solutions currently or previously in use proves that there is a requirement to improve device safety and implement new security techniques such as those that are recommended by this paper. The paper suggests that a cryptographic logging system be implemented into the device to allow for auditing via remote connection as well as an intrusion prevention system to prevent malicious device variables being passed to the pacemaker via the remote connection. The Paper makes suggestions that are also implemented into the prototype pacemaker and have been used due to current security concerns surrounding pacemakers.

Keywords: Pacemaker, Raspberry Pi, Intrusion Detection, Intrusion Prevention, Cryptographic logging

1. INTRODUCTION

The Pacemaker is a revolutionary design however it has undergone little change in its vast history, the Engineering and technology history wiki (Geselowitz & Leder, 2017) provides an understanding on how the device has been adapted over the years. However, the initial device was invented before the digital revolution which started circa 1975, devices are progressively being converted to 'smart' devices which allow for them to be connected via the internet and be classes as internet of things (IOT) Devices. Recent development on pacemaker technology by St. Jude Medical has led to the at home solution branded as the Merlin@home device is

capable of a few key features: Daily Monitoring, which allows the technician to review the patient's overall health; Direct Alerts, which send updates about a potential episode and event and communicates this information via Email and mobile apps; remote monitoring, which allows for scheduled uploads of patient data. However, this specific device was argued to be full of potential security flaws by investment firm Muddy Waters and is discussed in the literature review. However, it does pose the ultimate question of this paper which is to understand and develop solutions to potential flaws surrounding implantable pacemakers that have IOT functionality. Implementing security into such small lightweight devices can be hindered by factors such as storage, processing power and battery life however with constant improvements in all three of these sections implementation of a powerful security system can be added into a prototype system. For example, the Neuralink device which is intended to allow for man and machine to symbiotically exist together is stated by the company's CEO Elon Musk to use wireless charging and can be seen in fig 1.1.



FIG 1.1 – WIRELESS CHARGING

2. BACKGROUND

Ever since 2019 I have progressively gained interest into implantable computational devices such as RFID chips, Brain Implants and Pacemakers. This interest has led me to dedicate my final year of university to investigating and researching security flaws as well as these devices capabilities. A previous paper I have written during my final year discusses the security issues around integrated devices in insecure environments (Brook, 2020). This information has helped me discover more surrounding security with these devices such as a previous American vice president opting to have their pacemaker deactivated during their time within the white house in fear of potential assassination attempts via the device. The Research paper also allowed me to survey a group of students at Staffordshire university to investigate the average person's concerns around these devices and how many students are personally affected by them.

The Development of the computer brain interface chip, which is made by Neuralink sporting Elon Musk as their CEO helps show how implanted devices can be capable of extreme computational tasks even under size constraints. This set me onto a path of wanting to investigate current pacemaker technology to help design security solutions to improve the quality of device safety based on both previously stated issues and novel security systems.

3. FUNCTIONAL REQUIREMENTS

The Prototype pacemaker has a few requirements for it to be a useful prototype, The Moscow diagram fig 3.1 shows these features as well as those that are planned for later versions of the project.

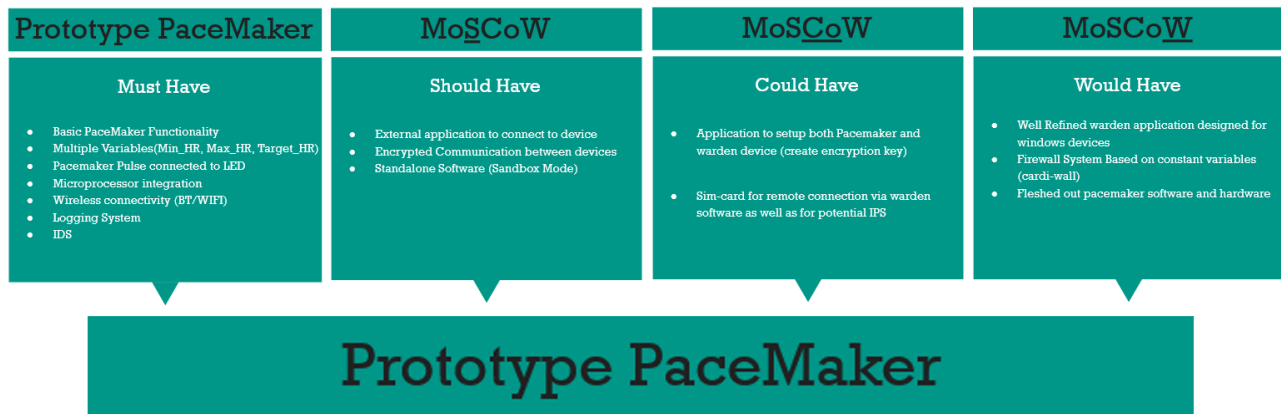


FIG 3.1 – MOSCOW DIAGRAM

The Prototype artifact must incorporate basic pacemaker functionality to allow it to detect the heart rate of the patient (Spoofed Heart Rate), the pacemaker must also be capable of outputting a pulse (Flashing an LED) to the patient. Due to many current pacemakers lacking connectivity this prototype must implement Wireless connectivity whether via Bluetooth or Wi-Fi. The prototype pacemaker must also implement an intrusion detection system based around a cryptographic logging system to ensure logs are confidential and any tampering will be detectable. The Prototype solution will also be developed to work on a Small Microcomputer such as an Arduino or other similar device. The pacemaker system will also revolve around set variables of Minimum Heart Rate, Maximum Heart Rate and Target Heart Rate.

The Pacemaker prototype should incorporate encrypted communication between device and client, this will ensure that all data transmitted can only be read by the intended users and improves the confidentiality of any medical information surrounding the patient as well as passwords entered into the system. An external application such as a client should also be developed to communicate with the prototype device to change key variables such as target heart rate as well as add new doctors to the system. The software should be capable of operating without the need of a microprocessor computer in a sandbox state.

The Development could implement a software that pairs the pacemaker and client application via generating an encryption key which will be used for secure communication and log cryptography. The prototype system could incorporate global communication via implementing a sim card slot which would allow for doctors and technicians to make modifications and check up on pacemaker at any time.

If the prototype solution was developed further into an official product, then a few elements would be added to it, the first being a professional refined software for windows devices to allow for technician training on this device to be minimised due to ease of use. A firewall system based on cardi-wall which is mentioned within the literature review and prevents malicious/accidental modifications to key variables within the pacemaker. The Pacemaker software will also be fleshed out to provide a better ease of use like incorporating a help command and other graphical updates.

4. PROJECT MANAGEMENT

Managing and planning the development and research for this paper allows for a more efficient workflow to be carried out. Planning out a rough path for the research and development assists with time management, which is extremely important when working on multiple modules at the same time. Not only can breaks be set up to dedicate time to other modules but the time management is a necessity as during the writing of this paper the Corona virus pandemic caused the work-life balance to change due to using the same area to work and live in. This new style of living added an extra challenge as my home computer is normally a space for entertainment whereas the library or local coffee shop is where I go to get work done. With this removed I needed to overcompensate on work to ensure that I had earned the right to relax in the same environment I was previously working. To implement this time management a Gantt chart was designed to show how the project would be carried out (fig 4.1). The figure shows how the project was set up to use all the available time spanning from October 2020 to May 2021. The Project is split into 4 large sections and 2 smaller ones towards the end of the project: Project Conception, consists of key elements required for writing a paper at Staffordshire university; Artefact Planning, allowed for an understanding as to what the artifact will be and how the lit review will be used to develop it; Literature Review, this section provides information on what topics will be reviewed to develop and build upon previous solutions and security issues.

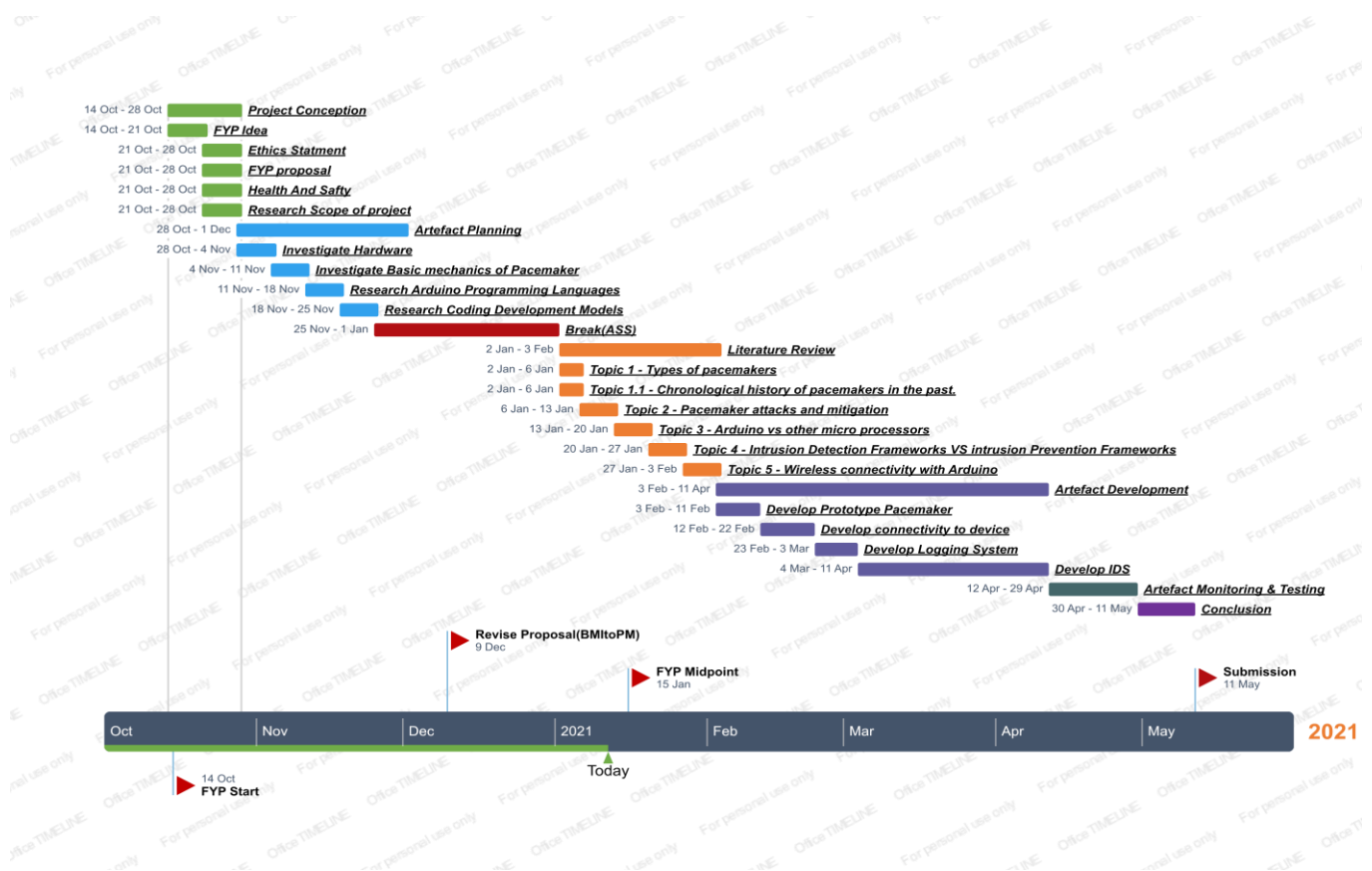


FIG 4.1 – PROJECT GANTT CHART

The smaller sections such as the artifact monitoring is used to ensure that the artifact is achieving the required features stated in the Moscow diagram above and is running correctly, the conclusion allows for the paper to be summarized showing what information was established from the literature review and development of the artifact.

5. LIT REVIEW

5.1 Pacemakers

Pacemakers play an important part within modern medicine and have come a long way since their early development to counter the risks of domestic electricity, this of course being suggested by (Geselowitz & Leder, 2017). Pacemakers are much more advanced than the original crude device called the “hymenopter”. Pacemakers later became portable with the use of portable electronics in 1957 (C. Walton Lillehei, et al., 1960). These devices were once again enhanced when (Greatbatch, 1962) published his patent for an implantable pacemaker. From this point onwards these devices inhibited advancements from each decade, which propelled the devices abilities.

Current devices can assist with multiple types of cardiovascular issues and can be seen in fig 5.1. The three types of pacemakers currently used within the medical industry are the following: Single-chamber, connects to the right ventricle or right atrium; Dual-chamber, connects to both the right atrium and the right ventricle; Biventricular, connects to the right atrium and both ventricles.

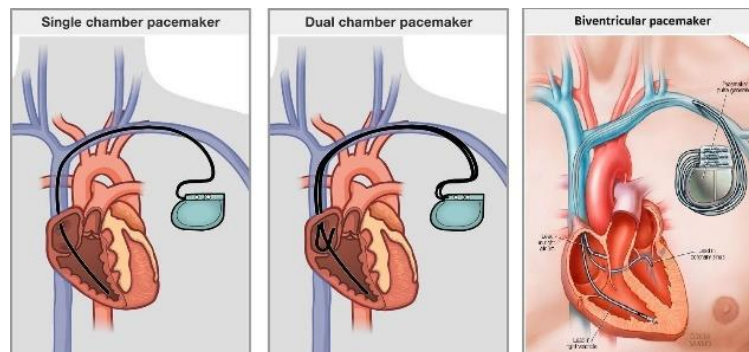


FIG 5.1 – TYPES OF PACEMAKERS

These modern versions are still being developed, (Awan, et al., 2018) suggests that the next generation of pacemakers will be entirely wireless. This would lead to a better accessibility to the devices enabling the professional’s quick access and auditing solutions to these devices. A serious problem arises when this wireless capability is implemented into the devices, that being unwarranted connections from potentially malicious third-party devices and users. (Geselowitz & Leder, 2017) suggests the initial development of pacemakers focused on not only starting and maintaining the pulse of the heart but also being able to stop the heart in order to force a reset upon it. This ability to reset the heart can cause a serious issue if only the stop command is outputted by the device the patient could be killed instantly, which would cause a serious outcry into how secure these devices are. A former VP of the United States had doctors purposely disable the pacemaker while he remained in office to prevent an assassination attempt on his life (ref?). The ability to murder someone with the use of a medical device designed to keep them alive is a worrying construct, to elevate this issue even more a paper written by (Fu & Blum, 2014) suggests that there is little to no auditing capabilities of these devices. Integrating intrusion detection into a device like this is a difficult task as (Davis, et al., 2019) suggests that power consumption is a large issue when developing an adequate solution.

A recent affair surrounding the Security and integrity of Merlin@home manufacturer St. Jude medical and investment firm muddy water shows that these devices may be riddled with vulnerable and buggy software which could cause potential data breaches or even result in the death of a patient.

This issue surrounding the Merlin@home system is heavily disputed by its manufacturer as they state, “the bugs were from a pre patched version of the software” (ref ?). To support this claim by the company and article stated that the firm muddy waters would not provide a comment. Furthermore, the fact that the software was ever public in a bug riddled state does not lead people to trust this type of device and this can be seen by that drastic plunge of 5% of St. Jude medical shortly after muddy water published their findings.

5.2 Attack Mitigation

Firewalls tend to be intrusion prevention systems (IPS) however they can also be used to enhance the detecting capability of an intrusion detection systems (IDS), as they are capable of logging network activity. The use of a firewall is to enforce a rule set that typically revolves around ports and addresses, but this is just the normal method of implementation. Implementing any IPS requires a slightly higher level of computational power when compared to an IDS as the IPS will need to scan every packet entering being transmitted to and from the device.

A paper written about a prototype firewall system called CardiWall (Kintzlinger, et al., 2020) suggests that a firewall could be implemented for use with these medical devices to secure pre-set variables from being altered by malicious attackers and even prevent accidental variable changes outside of a safe value. Fig 5.2 shows how the firewall can prevent the previously mentioned vulnerabilities from being attacked. This system may still require a large amount of processing power which could drastically deteriorate the battery's life which is normally around 10-15 years (Thamilarasu, et al., 2020). Due to the increased requirements to have this system on the implanted device the writer of CardiWall shows that the system is not to be implemented on the device instead it is implemented on the programmer's device to prevent them from altering the variables outside the safe operating area.

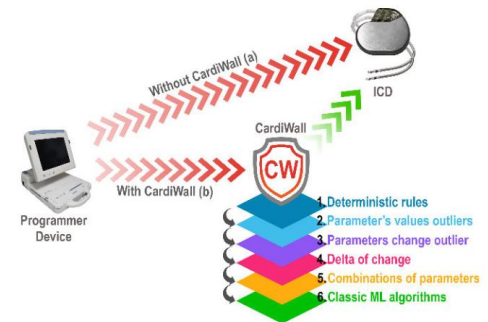


FIG 5.2 – CARDIWALL SYSTEM

Incorporating this type of firewall into the prototype system would require a secondary program to be built to monitor how the variables are being edited on implantable device, however the CardiWall solution will only defend against the programmer's but will leave the device vulnerable to attackers that are able to remotely connect to the devices outside of the "Safe-Environment" (Schneier, et al., 2002). If this was to be implemented into the prototype power consumption and CPU usage should be closely monitored to prevent the device requiring more frequent battery change operations.

5.3 Microprocessors



FIG 5.3 - MKR NB 1500



FIG 5.4 - UNO REV3



FIG 5.5 - NANO 33 BLE

(Arduino, 2018) states that they have built up a community of developers ranging from novices all the way to professionals. The information later goes on to state how these groups together have amassed a large amount of knowledge that can be found from the site Arduino.cc, Information of course can also be found on the internet and in books.

The Arduino board's appearance was in 2005 (O'Reilly Media, Inc., 2016). The devices seen since the initial development have been designed to fit into specific needs and requirements of the groups that were mentioned within (Arduino, 2018). It is also mentioned within (O'Reilly Media, Inc., 2016) that the devices range in size, with some being larger than the original and others being smaller.

The Arduino can be bought in kits, which come preloaded with additional modules for the main board. These kits increase the price however can allow for endless possibilities. It is stated that a standalone board costs around \$30 and a kit costing double that at \$60 (Barrett, 2013). The kit form of the Arduino is slightly pointless when sites like rapid online (Rapid Electronics Limited, 2021) which allow for the exact components to be purchased. Even with this being said the kits all consist of identical components which allows for Arduino libraries to be adapted into the desired application with ease, instead of having to find the manual on each bespoke electronic component.

As mentioned previously in the intro to Arduinos the size can vary for each device, (Arduino, 2021) allows for these different devices to be purchased. Fig 5.3 shows a board that is about 68x25mm and weights around 32g. The second Arduino fig 5.4 is a slightly larger model which is around 69x54mm and weighs 25g. The final Arduino seen fig 5.5 is around 45x18mm and 5g which lends itself to be one of the smallest boards in the Arduino family.

However, the smaller form factor devices may require additional setup, the board seen in fig 5.5 requires the pins to either be soldered to in the case of long-term use, or the boards pins to be inserted into a breadboard. Whereas the other two boards are standalone and can have electronics slotted into the pinholes provided. Being able to get variable sized boards is also possible when using the raspberry pi which can be seen in its subsequent section.

The Arduino comes equipped with 32kb of memory for storing programs, this memory size is quite small when compared to an everyday computer which tends to have around 250gb+. Even with this miniscule amount of data fig 5.6. This obviously puts a cap on project size as the memory cannot be easily increased by adding a new SD card which is a feature available on Raspberry Pi putting the Arduino at a slight disadvantage when it comes to this. Even with this being said the storage size really isn't an issue if the practical application is very specific and does not require a massive program which requires more than 32kb of storage. The size of programs has gradually been getting larger as the storage size of devices change, they also get large due to lazy programming which is called software bloat which tends to be from feature creep and is suggested by (Jiang, 2017) and mitigations to this are also mentioned which can show how some bloating can be removed.

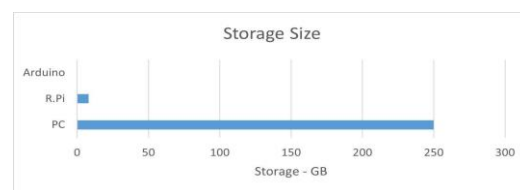


FIG 5.6 – STORAGE SIZE

The processing speed of a device is extremely important for how many instructions can be carried out per second. (Arduino, 2021) provides information on the technical specifications for the Arduino chips and states that its clock speed is 13MHz, this once again like the storage size is drastically small compared to an everyday computer using an i5 processor and is still drastically low compared to the Raspberry Pi the comparison can be seen in fig 5.7.

The ability to implement the board into the specific project is made possible with the size mentioned above and also the modular ability. The Arduino is designed to be incorporated with small electrical projects that consist of sensors, this is mentioned in the book Arduino Projects to save the (Premeaux & Evans, 2011). The book also goes into detail about how the device can accept digital and analogue signals, which allows it to fit into multiple types of projects. The Arduino can be modified using extra modules which are available at multiple sellers and can also be seen at (Arduino, 2021) requiring more modules can also force the price of the device to go up, whereas the raspberry Pi can allow for modular components to be added but contains a lot of built-in functionality and can be seen in (Raspberry Pi, 2021).

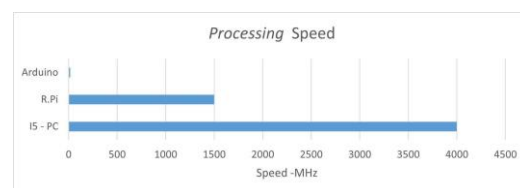


FIG 5.7 – PROCESSING SPEED

Arduino pricing varies from each different board, looking at their store page (Arduino, 2021) shows that the boards that have more functionalities built into them tend to be higher priced, but this price range is around £8-£80. For this project, the Arduino Uno r3 has been chosen as the top Arduino to be used due to the price only being around £18 this is £16 less than the latest raspberry pi of roughly the same dimensions.

The Raspberry Pi is a miniscule computer, unlike the Arduino this device is capable of running multiple operating systems, that can be seen on more traditional computers, for example Linux mint. This device was designed to provide everyone with computational power and can be seen in (Raspberry Pi, 2021). The Raspberry Pi did start with a similar design to the Arduino as its first conceptional idea circa 2006 was to allow for the user to build it themselves using common electrical components.

When it was first revealed in 2011 the device had changed to be a microcomputer that was pre-assembled (Heath, 2018). This device would allow for multiple coding languages to be used when implementing the prototype design for the pacemaker and its intrusion detection system.



FIG 5.8 - RASPBERRY PI ZERO W



FIG 5.9 - RASPBERRY PI 4 B

The Raspberry Pi is available in multiple versions, these devices are mainly the same size of around 85.6x56.5mm. The company offers a smaller version called the Raspberry Pi Zero W seen in fig 5.8 which clocks in at 65x30mm (SocialCompare, 2021) which is an almost identical size to the MKR NB 1500 offered by Arduino and can be seen in the previous section.

The Raspberry Pi on its own has no real storage, the device instead opts for a SD card to be used which allows for the device to have a range of storage capacities. (Raspberry Pi, 2021) suggests that a minimum size of 16 should be used with the Raspberry Pi's NOOBS Operating System. This reference also states that the device is capable of reaching >64gb when the SD is formatted in exFAT. This storage capacity Leaves ample room for the prototype system to be coded.

The Processing Speed Varies between the two previously mentioned devices with the larger Latest Raspberry Pi fig 5.9 a 1.5GHz processor and the Raspberry Pi Zero W fig 5.8 having only 1GHz. This processing power is still much higher than that offered by the Arduino board which sits at 1.3% of the Smaller Raspberry Pi Board.

Both boards mentioned above provide ample room for adaptation into the prototype device that will be designed, these boards both have pre implemented Bluetooth and wireless functionality (Raspberry Pi, 2021) which the Arduino Rev3 does not have however the smallest Arduino board does have Bluetooth connectivity.

Due to the Raspberry Pi being a more flexible system and providing extra features, the price of the device seen in fig 5.9 is much higher than that of the similar sized Arduino model and sits at around £34, the smaller Zero board seen in fig 5.8 is roughly half the price at £9.30 of the Arduino board and offers much more functionality however it does require a power supply and SD card for it to function which will add around £10 to the price.

Reviewing the following information about how each board could be implemented into the prototyping of the pacemaker, as well as understanding their advantages and disadvantages over each other like price, processing speed, storage capacity and inbuilt capability such as connectivity it is clear that the raspberry pi zero W board can offer much more than the Arduinos, in all regions of comparison that would be required for a valid selection process.

5.4 Intrusion System

The two types of intrusion systems both have advantages and disadvantages however the intrusion detection system offers a large amount of security at a lower CPU cost which is important to consider for when the device will be implemented as the battery life of pacemakers is around 15 years and eating into this can cause potential surgery complications to occur, however with battery technology improving and development into recharging implant devices such as Neuralink the battery life may be able to be made exponentially shorter allowing for more intensive software to be run. A concept design for charging a brain implant can be seen in 5.10 however this is not an official device. If there is potentially extra development time due to predicted project times being less the intrusion prevention system will be implemented on top of the IDS.



FIG 5.10 IMPLANT CHARGER

5.5 Connectivity

The prototype device that will be designed should have some method of connectivity to allow for variables to be set by the doctors/programmers. Currently the more prominent companies developing pacemakers are working towards wireless connectivity with these devices as any method that involves physical connection means that the patient's chest cavity will need to be opened or a plug will be protruding on the surface of the body. Connecting these devices to the internet is stated to provide advancements in monitoring as devices could be checked in real time and store log to the cloud for doctors to review and understand how the patient's heart is working with the implanted device (University of the Basque Country, 2015). Currently St. Jude Medical has a Radio Frequency device that is capable of communicating with the device while they are in close proximity. This device allows for the user to be alerted while they are sleeping if the device has detected anything, the device is called Merlin@home. The system allows for the communication of pacemaker data to be transmitted from this external device to allow for remote monitoring, this remote monitoring is only possible while the device is in close proximity though (Abbott, 2021). Both selected prototype boards have the ability to implement connectivity.

Bluetooth and wireless are almost identical methods of connectivity both using forms of radio bands to enable communication between devices. A book written by Matthew Gast (Gast, 2005) goes into detail on how wireless systems provide multiple improvements for office environments however he also states the use of flexibility which, in this case can be implemented into the pacemaker allowing it to connect to multiple devices allowing for a constant connectivity to be maintained. This also provides the possibility of being connected to a mobile phone and switching to a home base station or even allowing the device to automatically connect to the closest secure device like a smart watch. The paper written by (Haartsen, 2000) discusses how Bluetooth radio transmitter implementation in microelectronics is increasing constantly, this shows that companies are opting into using Bluetooth connectivity to most portable electronic devices.

The use of a wired form of connection of an implanted device poses a few serious concerns for the patient, one the most serious ones being risk of infection from having a seemingly open wound into the chest cavity. This issue means that wires that extend out of the body should not be used.

5.6 Encryption

The use of encryption has been used since messages needed confidentiality to ensure only specific individuals would be able to understand the transferred information and has been around for a long time. Current systems are still being aided by the book called Kautilya-Arthaśāstra (कौटिल्य-अर्थशास्त्र) which is from circa 300 BCE (Shamasastri, 1956). Transmitting data from one device to another as well as storing data can allow for vulnerabilities to appear. To mitigate against the issues that arise, encryption may be used, it provides ample ability to hide and distort the original data that was stored or transmitted. A paper written about intrusion detection systems (Thamilarasu, et al., 2020) suggests that encryption requires a large amount of computational power and could cause issues as a pacemaker is a portable device with a typical battery duration of 5-15 years (Davis, et al., 2019). Other than computational restraints implementation of these devices can provide a high level of security for a small overhead with a relatively ease of implementation once ample CPU power and battery

capacity is used. This minimum level of security can provide a fairly safe environment and prevents critical data being leaked about the patient.

Encryption is not just a blanketed fix as there are two types of encryption, both with large pros and cons as well as many ways of implementing them. There is symmetric encryption which uses the same key to encrypt and decrypt which can be seen in action when communication to web servers takes place and the SSL lock appears in the search bar 5.11. The prominent method for establishing a symmetric system revolves around Diffie Hellman an asymmetric system and is talked about below. An algorithm that specifically used symmetric encryption is TwoFish which was an AES candidate.



FIG 5.11 – SSL LOCK

The other method of encryption is asymmetric which involves the use of two keys, a public and private key. The sender would encrypt their message using the receiver's public key, which could then only be decrypted using the receiver's private key. This provides confidentiality as the only one that can read the message is the receiver as no one else has their private key. Asymmetric encryption is much slower than its counterpart as unlike symmetric encryption it was not designed to encrypt and send mass amounts of data at breakneck speeds. Instead, Symmetric encryption can provide a slightly higher level of security that can be seen in the Digital Signature Algorithm (DSA) which is a member of the Digital Signature Standards (DSS).

5.6.1 TWOFISH

This encryption system was designed around 1998, to compete to be the next Advanced Encryption Standard (AES) (Schneier, et al., 2002), the encryption process revolves around a block cipher that uses 128 bits. This works via segmenting the data being processed through the encryption system. This AES candidate lost to the more efficient Rijndael system as it was proven to be more efficient on devices lacking processing power (Joan, 2010). The method of encryption is a symmetrical system requiring the same key to encrypt and decrypt.

5.6.2 DIFFIE-HELLMAN

This system was designed to create a secure line of communication for a symmetric encryption key to be transferred as these types of systems are much faster for processing and transferring data. This is made possible due to large primes being calculated to obtain their modulus (Hellman, et al., 1977). Looking at fig 5.12 the use of this modulus can be seen as the public keys are transferred to create a shared private key.

User1: $K_1 = G^{r1} \text{ mod } P$	K_1 = User 1 Public Key
User2: $K_2 = G^{r2} \text{ mod } P$	K_2 = User 2 Public Key
User 1: $k = K_2^{r1} \text{ mod } P$	k = Private Key
User2: $k = K_1^{r2} \text{ mod } P$	P = Prime #
	G = Large Int
	$r1$ = User1 large ran int
	$r2$ = User2 large ran int

FIG 5.12 – DIFFIE-HELLMAN

5.6.3 AES

Advanced Encryption Standard is currently approved under the FIPS for securing data, the algorithm assigned to the AES is called Rijndael (National Institute of Standards and Technology, 2001). This system just like TwoFish is based on a block cipher which allows key sizes of 128,192 and 256 bits with a 128bit block. This encryption standard was implemented on May 26th, 2002.

5.6.4 DSA

The use of the digital signature algorithm is to append a form of integrity to the message being transferred. The system uses both encryption and one way hashing functions to provide both security and integrity, the use of the has proven that the file has not been altered during transit, this algorithm was first published in 1991 for the FIPS (Kravitz, 1991). This system could be used with Sha-2 hashing as well as the current Advanced Encryption Standard mentioned above.

5.7 Language

Selecting a language for this prototype is important as for it to be efficient it will need to comply with two things, being available on the device that is to be used as well as fall under previously used coding languages. The requirement for it to be previously used means that development time can be spent developing features rather than developing and understanding on how to use the language, this can provide more advanced features and also decrease the size of the final program as bloatware can be avoided. Fig 5.13 shows the language compatibility with devices and this project.

Languages	Raspberry Pi	Arduino	Computer	Experience
C#	Yes	No	Yes	Yes
Python	Yes	No	Yes	Yes
Java	Yes	No	Yes	No
Java Script	Yes	No	Yes	No
PHP	Yes	No	Yes	No
Ruby	Yes	No	Yes	No
C++	Yes	Yes	Yes	Yes
C	Yes	Yes	Yes	No

FIG 5.13 – LANGUAGE COMPARISON

5.7.1 THE LEARNING CURVE

As mentioned before using a language which has been used for previous projects allows for part of the learning curve to be removed, saving time that would otherwise be spent understanding the basics of the language. The paper written by (Morrison, 2008) discusses how the learning curve theory shows that the more time spent on learning something the quicker the new skills can be developed as the productivity improves, this term is labelled cumulative production. The paper from (Morrison, 2008) also states that the model is designed out of repetition, one of the more natural ways of learning something and can be compared to learning to walk as this process will lead to the person falling multiple times until eventually, they are capable of taking steps. The model in fig 5.14 uses the fact that people will forget things and incorporate this into the model, this is extremely important as even using a previously used language can still be susceptible to losses knowledge of tasks that are not commonly repeated.

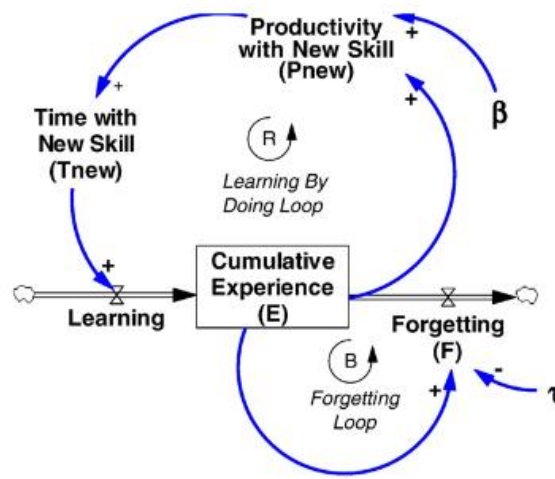


FIG 5.14 – LEARNING CURVE

5.7.2 LOW-LEVEL & HIGH-LEVEL

Languages normally fall into two groups, low-level and high-level. This determines how much is going on “behind the scenes”. From the programs listed below C++ is the lowest of the languages and can provide fine tuning of memory usage and allow for bloat code to be removed. Whereas python is a high-level language and sits on top of a line-by-line interpreter as well as allowing for code to be easy to read and could cause for unnecessary bloat code to be created if it is compiled/interpreted.

The importance of picking the correct language comes down to what the project will require on a technical stance. In this specific case a prototype system will be designed, prototyping will be discussed in more detail below however the premises involve rapid development to show a potential feature or system. Low Level languages are not necessarily required as the prototype system will likely not be built on the same system as the final product. The prototype pacemaker will be designed to work with a non-custom-built board with little to no memory restrictions or processing power. Which means the benefits of hand crafting elements with a low-level language will not be needed until the system is designed for a smaller board.

5.7.3 PICKING A LANGUAGE

Understanding which language to use will be determined by board as well as the amount of previous experience with the programming language, therefore a table (Fig L.3) has been designed to represent which language should be used based on a few principles, Projects made using the language: time using the language and the prototype ability of the language.

Language	Project #	Years Experience	Prorotype-ability
C#	2	5	High
Python	4	7	High
C++	1	2	Low

FIG 5.15 – LANGUAGE COMPARISON 2

The table shows that python is the top language to use due to the 7 years of experience with the language, the use of python is also corroborated by the paper written by Raphael valet (Vallat, 2018), as it states that the language has a fast learning curve which will help to jump back into the language as well as develop new skills. The paper also discusses that the language is extremely useful for statistical analysis when the package Pingouin is used, this open-source statistics package could assist with developing a better system as heart rate information and device logging could be modelled.

6. ANALYSIS

The literature review has looked at the following pieces of information that are essential for developing an ample solution in the form of a prototype pacemaker: Pacemaker information, this covers the history, vulnerability and types of pacemakers; Attack Mitigation & Intrusion System, these sections within the lit review look into the currently used solutions for preventing attacks as well as what could be implemented to assist with preventing them in the future; Microprocessors, reviews different types of hardware that could be used to develop the prototype solution on; Connectivity, is used to research information surrounding previous solutions methods of communication as well as the types of communication that could be implemented into the prototype solution; Encryption, the final section of the lit review focuses on 4 encryption methods that are currently in use around the world and reviews how their strengths make them a good candidate for being incorporated into the solution.

The final step before developing the artefact is to select which solutions from the literature review should be incorporated, to assist with this the Moscow diagram seen in fig 3.1 within the project management section will be referenced to ensure that all the requirements are met. Previously mentioned the Type of microprocessor has already been selected as the raspberry pi zero wh as it is capable of offering a large number of features and flexibility to the project. The literature review also explains how Bluetooth is largely used in small portable electronics however due to ease of implementation Wi-Fi connection will be used for the connectivity of the prototype pacemaker. As mentioned above the paper reviews 4 encryption methods and from all of these the AES method is best suited for the prototype pacemaker due to the large number of companies and countries that adopt this encryption system as well as the ability to incorporate the system via use of the pycrypto library for python. The Final element to consider is the intrusion system, the Moscow diagram fig 3.1 states that there should be at least an intrusion detection system in the form of a cryptographic log, however the addition

of an intrusion prevention system similar to the system called CardiWall seen above will also be implemented into the prototype solution.

7. ARTEFACT DEVELOPMENT

7.1 Design

The design for this artefact takes into account the conclusions made from the literature review, it also will build upon those ideas to make an abstract version of what is required of the program. This abstract version can be seen in a class diagram in 6.1. The class diagram shows that the program will be split into several sections: Heart Rate Detection, Pacemaker Connection, Pacemaker Log, Pacemaker System Admin, Pacemaker Firewall and Pacemaker Main. Each of these components plays a key part and will be discussed in order of which should be implemented first. The pacemaker main section is the brain of the program, it will allow for comparisons of the data collected with the heart rate detection to be compared with predefined values. The Firewall part of the program is used to set up constant values to prevent anyone who gains access to the system from tampering the device beyond safe operating conditions which was discussed previously within the attack mitigation section of the literature review. To allow for the Pacemaker main variables to be modified the system admin section is designed to allow for connected users to edit variables within the range that the firewall has allowed. Implementing connection to the device is the next process and allows for the external user access to the device even when it is implanted in the body, all communication between the server and client will be logged by the log class.

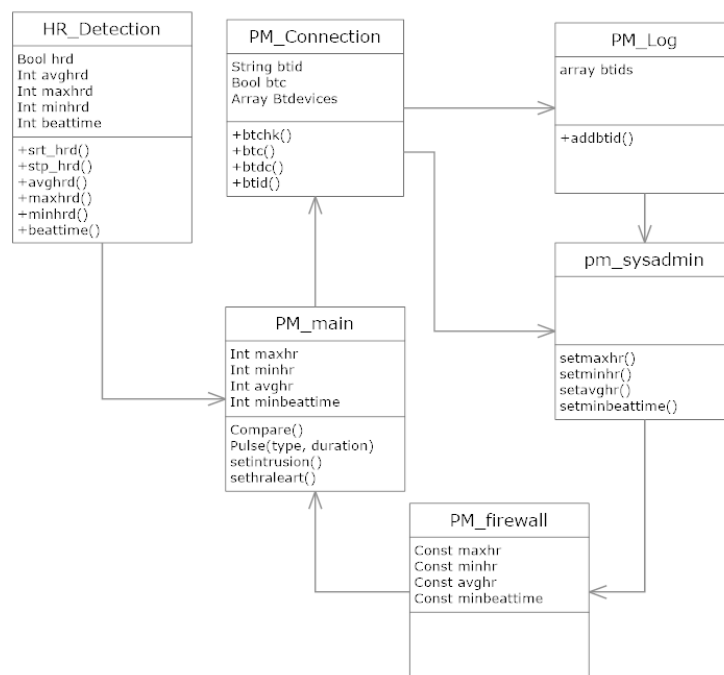


FIG 6.1 – CLASS DIAGRAM

7.2 Development & Implementation

The first step of development is to create a working environment, this is split into two sections the software aspect and the hardware side. There are 2 key services that are used: The Integrated development environment (IDE), which is designed for python and called PyCharm; The storage service, which allows for version history to be saved and for ease when setting up the code on multiple devices called GitHub.

The Prototype Pacemaker is designed using the programming language python which unlike other languages uses an interpreter to read the code line by line. The language python allows for programming to be quick which allows for it to fit into the small time slot given to the artifact development in the Gantt diagram, due to it being such a high-level language. The PyCharm IDE includes as implementation of GitHub which allows for the code to be pushed from within the software and allowing for more versions to be made, this can be seen in the fig 7.2 this figure also shows the environment that the program will be coded in. To Implement GitHub into the development the folder must first be initialized. Once initialized the git repository can be pulled to a local version and then once adequate changes have been made the new version can be pushed back to the repository. There is vast documentation explaining how to use GitHub and setup projects both locally and on their servers this can all be found on their website ([GIT HUB DOCUMENTATION](#)).

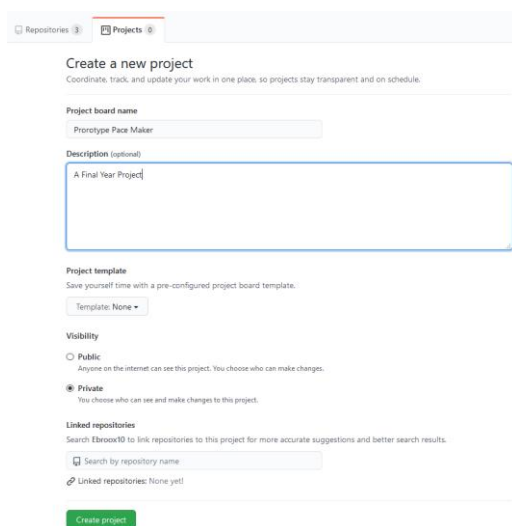


FIG 7.1 – GITHUB SETUP

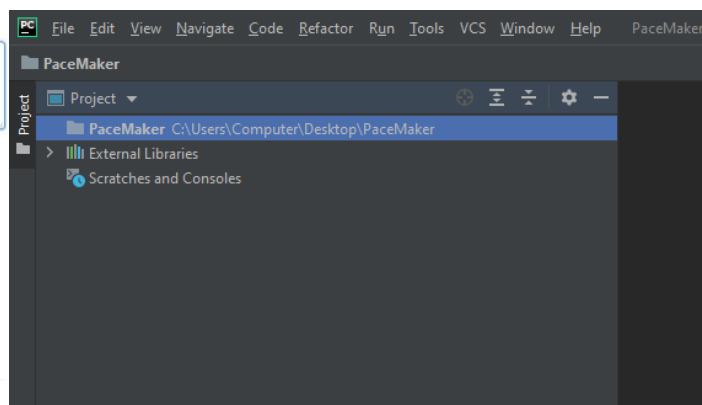


FIG 7.2 PYCHARM SETUP

Hardware development revolves around a few LEDs, wires, a bread board and a Raspberry pie Zero WH which was discussed earlier. The Raspberry pi is running a headless command line version of Linux called raspberry pi OS lite, which can be connected via SSH to pull the latest version from within the GitHub repository.

Development of the artefact was split into 4 key sections: Pacemaker Development, focused on creating fundamental software for a pacemaker; Device Connectivity, consisted of implementing client-server communication for editing variables; Logging System, incorporated a log system into the client-server communication and alerts made by the base program; IDS development focuses on converting the log system into a cryptographic log system capable of encrypting data using AES.

7.2.1 THE BASE PACEMAKER

The Base software for the pacemaker was given a few requirements from the Moscow which can be seen above. Achieving these requirements was possible by coding two classes The Main Class and the Heart Class. The Heart class was designed due to the prototype nature of this artefact and allows for heart data to be spoofed from within the application. This class consists of two functions the bps () fig 7.3 which calculates the beats per second of the heart and the beat() function 7.4 which is used to output a pulse to the heart led in sync with the heartbeat.

```
def bps():
    if Heart.hearthealth == 0:
        Heart.rantime = random.uniform(1, 1)
    if Heart.hearthealth == 1:
        Heart.rantime = random.uniform(0.95, 1.10)
    if Heart.hearthealth == 2:
        Heart.rantime = random.uniform(0.95, 1.20)
    if Heart.hearthealth == 3:
        Heart.rantime = random.uniform(2, 2)
    if Heart.hearthealth == 4:
        Heart.rantime = random.uniform(0.90, 1.5)

    Heart.bpm = (60/Heart.rantime)
```

FIG 7.3 – BPS FUNCTION

```
def beat():
    sleep(Heart.rantime)
    #print("SHEARTBEAT")
    #GPIO.output(OHEART, GPIO.HIGH)
    #GPIO.output(PHEART, GPIO.HIGH)
    #sleep(0.1)
    #GPIO.output(OHEART, GPIO.LOW)
    #GPIO.output(PHEART, GPIO.LOW)
```

FIG 7.4 – BEAT FUNCTION

The BPS function uses the heart health variable to determine which pre-set of random heart rates should be used, each heart health has a range of beats per second and allows for a realistic heart to be simulated and provides the software to be tested in a “sandbox state”. The function continues to use this variable converts it to beats per minute which is then stored in the classes’ bpm variable.

The beat function is part of three outputs that are incorporated into this artefact, this specific output is possible due to the Raspberry Pi’s inbuilt GPIO pins and the python library called Rpi.GPIO that they have developed. Setting up the GPIO pin to be an output is extremely simple as it consists of setting the pin to be an output, followed by setting a pin value to an easily readable variable and setting that output pin to high or low fig 7.5. This section of code is commented out as the GPIO functionality is not possible if the application is running on a system other than a raspberry pi, however when implemented on a raspberry pi the output is used to light up a red (Heart Rate) LED.

```
9 #GPIO stuff
10 #GPIO.setmode(GPIO.BOARD)
11 #GPIO.setwarnings(False)
12
13 #PULSE = 35
14 #OHEART = 37
15 #PHEART = 33
16
17 #GPIO.setup(PULSE, GPIO.OUT)
18 #GPIO.setup(OHEART, GPIO.OUT)
19 #GPIO.setup(PHEART, GPIO.OUT)
20 #GPIO END
```

FIG 7.5 – GPIO SETUP

The Main Class Consists of 3 functions which make up the base mechanics of the pacemaker, these functions are capable of comparing, calculating and alerting information to the system. The pm() function is the first function called by the program via threading. This is important for later versions of the software that implement connectivity as the client will be able to connect even while the pacemaker is detecting and calculating information about the heart. The implementation of threading is made possible with the library called threading which is incorporated in the base installation of python 3.9, the use of this library can be seen in fig 7.6.

```
121 ppm_main = Main()
122 Thread(target=ppm_main.pm).start()
```

FIG 7.6 – THREADING

The alert () function is used to detect if the heart rate generated by the Heart class is below or above pre-set variables by the user, if any variation is detected by this system it will output to the server and log the alert along with a timestamp of when the alert occurred fig 7.6. If the heart rate is within the predefined range then the function will return without outputting any data to the log.

```
def alert(self):
    if (Heart.bpm < self.min_hr):
        print(f"{datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')}: !!!!!Heart Rate LOW!!!!!")
        LC.log(f"{datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')}: !!!!!Heart Rate LOW!!!!!")
    if (Heart.bpm > self.max_hr):
        print(f"{datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')}: !!!!!Heart Rate HIGH!!!!!")
        LC.log(f"{datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')}: !!!!!Heart Rate HIGH!!!!!")
    else:
        return
```

FIG 7.6 – THREADING

Determining the average heart rate over 5 second intervals is carried out by the avgcal() function. The function will add 5 detected heart rates to the hrlist[] and then use this information to calculate an average (hrlist[] / 5) and store this information in the avghr variable within the Main class.

The pm() function's first step is to call the Heart.bps() function which is mentioned above, then call both the avgcal() and alert() function. The pm() function then proceeds to determine if the heart rate is below the target heart rate via a simple comparison, if the detected heart rate is less than the target heart rate the program will issue a pulse to match the predetermined target heart rate. This pulse is visualised via using the GPIO output and connecting it to a blue (Pulse) LED. If the detected heart rate is not less than the target the heartbeat is outputted, and the red (Heart Rate) LED is flashed fig 7.8. This function is also part of outputting to the pheart GPIO which is used to show how the heart is beating once connected to the pacemaker, it beats in sync with both the heart rate and the pulse output.

```
def pm(self):
    while True:
        #print("-----")
        Heart.bps()
        self.avgcal()
        self.alert()
        if (Heart.bpm/60) < (self.set_hr/60):
            #Thread(target=Heart.beat()).start()
            sleep(60/Main.set_hr)
            print(f"{datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')}: !!!!!Pulse!!!!")
            LC.log(f"{datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')}: !!!!!Pulse!!!!")
            #GPIO.output(PULSE, GPIO.HIGH)
            #GPIO.output(PHEART, GPIO.HIGH)
            #sleep(0.1)
            #GPIO.output(PULSE, GPIO.LOW)
            #GPIO.output(PHEART, GPIO.LOW)

        else:
            sleep(60/Heart.bpm)
            print(f"{datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')}: !!!!!HeartBeat!!!!")
            #GPIO.output(OHEART, GPIO.HIGH)
            #sleep(0.1)
            #GPIO.output(OHEART, GPIO.LOW)
        #print("-----")
```

FIG 7.8 – PM FUNCTION

```
def avgcal(self):
    #print("TEST")
    if self.count == 5:
        Main.avghr = (self.hrlist[0] + self.hrlist[1] + self.hrlist[2] + self.hrlist[3] + self.hrlist[4])/5
        #print (Main.hrlist)
        self.count = 0
        #print(Main.avghr)
    else:
        Main.hrlist[self.count] = Heart.bpm
        #print(self.hrlist)
        #print(self.count)
        self.count += 1
```

FIG 7.9 – AVGCAL FUNCTION

7.2.2 THE LOGGING SYSTEM

The Logging system is a relatively small section and only consists of one function `log()` fig 7.10. This system is designed to incorporate the intrusion detection system into it and is able to do this with the use of the custom AES encryption system which allows for the cryptographic log system to function. The log function allows for data to be imputed and then stamps the data with a number to show the order of the log as well as show if sections of the log have been removed or lost. The data is then fed into the ECB class and encrypted and outputted to the `log.txt` file as encrypted hex code fig 7.11.

```
1  from AES import ECB
2
3  class LC():
4      sfcnt = 0
5      epass = "9876"
6      def log(data):
7          logtxt = f"{LC.sfcnt}:{data}"
8          log = open("Log.txt", "a")
9          #print((ECB.encrypt(logtxt, epass)), file=log)
10         print(logtxt, file=log)
11         LC.sfcnt += 1
12         log.close()
```

FIG 7.10 – LOGGER SCRIPT

FIG 7.11 – LOG.TXT

The AES System uses a pre-built library called `pycrypto` and allows for SHA-256 hashing and AES encryption to be carried out with ease. This system is implemented into the ECB class which stands for electronic code book, the easier method of AES to implement. The ECB class consists of 3 important functions required for encrypting and decrypting and a 4 function which can be used as a user-friendly method of decrypting the log file. The `encrypt()` function fig 7.14 is responsible for encrypting data and takes to inputs the data and the encryption password, this password is then pushed through the `hash()` function which is responsible for creating the hashed password and generating an encryption key fig 7.12. After this section, the encryption method uses a fixed block size of 16 and a pad of the character `"{"` to ensure that the block size is always met. The function encrypts the data and padding and outputs it as hex code. The `decrypt()` function fig 7.15 uses the same inputs as the encryption function however the data is replaced with the encrypted hex code. The function converts the hex back to bytes and decrypts it using the password imputed to the function, the function then strips the padding from the decrypted text and returns the output as plain text. The final function called `start()` fig 7.14 requires 3 inputs, the mode(encryption or decryption), data(plain text or hex code) and the password.

```
def hash(password):
    phash = SHA256.new(password.encode('utf-1
    key = phash.digest()
    return key
```

FIG 7.12 – HASH FUNCTION

```
def start(mode, data, password):
    if mode == "e":
        #print("encrypting...")
        ECB.encrypt(data, password)
    if mode == "d":
        #print("decrypting...")
        ECB.decrypt(data, password)
    return ECB.ecbout
```

FIG 7.13 –START FUNCTION

```

def encrypt(data, passwd):
    key = ECB.hash(passwd)
    BLOCK_SIZE = 16
    PAD = "{"
    #padding = lambda s: s + (BLOCK_SIZE - len(s))* PAD
    padding = (BLOCK_SIZE - (len(data) % BLOCK_SIZE)) * PAD
    #print (len(data))
    data += padding
    #print(f"This is padding: |{padding}|")
    #input()
    cipher = AES.new(key, AES.MODE_ECB)
    ECB.ecbout = (cipher.encrypt(data.encode('utf-8'))).hex()

    #print ("encrypted hexcode:", ECB.ecbout)
    return ECB.ecbout

```

FIG 7.14 – ENCRYPT FUNCTION

```

def decrypt(data , passwd):
    #bdata = bytes(data, 'utf-8')
    bdata = bytes.fromhex(data)
    key = ECB.hash(passwd)
    PAD = "{"
    decipher = AES.new(key, AES.MODE_ECB)
    #print(bdata)
    ptext = decipher.decrypt(bdata).decode('utf-8')
    pad_index = ptext.find(PAD)
    result = ptext[:pad_index]
    #print ("decrypted text:", result)
    return result

```

FIG 7.15 – DECRYPT FUNCTION

7.2.3 SYSTEM ADMIN

The System Admin code uses two classes: SysAdmin, used for inputting commands to the pacemaker and outputting information; PaceWall, and Intrusion prevention system that prevents variables from being maliciously changed. This Function prevents the max heart rate, min heart rate and average heart rate from being changed out of a safe range via comparing it to pre-set variables within the system fig 7.16. Later implementation of this system may opt to use external variables that are not hard coded into the software such as a read-only file. The SysAdmin class consists of many functions each their own command, this class allows for commands to be implemented into the system in a time saving manner. The prototype device incorporates 9 user commands over 10 functions.

The First command seen is the set max heart rate command which is the function “setmaxhr()” fig 7.17 this allows for the user to change the maximum heart rate that can be reached before the base system will alert the log that the heart rate is too high. The second command is used to Set the minimum heart rate and is the “setminhr()” function fig 7.18 and allows the user to set the minimum heart rate before the system is alerted. The set average heart rate command seen as “setavghr()” allows for the target heart rate to be changed which will affect when the pacemaker pulses blue LED and can be seen in 7.19.

```

110 class PaceWall():
111     pw_maxhr = 100
112     pw_minhr = 40
113     pw_maxavghr = 90
114     pw_minavghr = 50
115
116     def maxhrchk():
117         if (SysAdmin.maxhr <= PaceWall.pw_maxhr):
118             Main.max_hr = SysAdmin.maxhr
119             return True
120
121         else:
122             return False
123
124
125     def minhrchk():
126         if (SysAdmin.minhr >= PaceWall.pw_minhr):
127             Main.min_hr = SysAdmin.minhr
128             return True
129
130         else:
131             return False
132
133     def avghrchk():
134         if (PaceWall.pw_minavghr <= SysAdmin.avghr <= PaceWall.pw_maxavghr):
135             Main.set_hr = SysAdmin.avghr
136             return True
137
138         else:
139             return False

```

FIG 7.16 – PACEWALL CLASS

```

def setmaxhr(option):
    SysAdmin.maxhr = option
    out = PaceWall.maxhrchk()
    return out

```

FIG 7.17 – SET MAX FUNCTION

```

def setminhr(option):
    SysAdmin.minhr = option
    out = PaceWall.minhrchk()
    return out

```

FIG 7.18 – SET MIN FUNCTION

```

def setavghr(option):
    SysAdmin.avghr = option
    out = PaceWall.avghrchk()
    return out

```

FIG 7.19 – SET AVG FUNCTION

The system also has commands that allow the user to see information about the system, this includes: `chkinfo` fig 7.22, this function allows for the system to output the min, max, avg and set heart rate; `chkuser` fig 7.20, allows for the user to see what users are currently added to the system by listing their usernames; `chkuptime` fig 7.21, is used to see how many days the system has been running for.

```
def chkusr():
    out = '|'.join(f'{key}' for key, value in SysAdmin.userdata.items())
    return out
```

FIG 7.20 – CHKUSR FUNCTION

```
def chkuptime():
    #out = f"Uptime is {}"
    out = f"Uptime: {(datetime.datetime.now() - (starttime)).days}"
    return out
```

FIG 7.21 – CHKUPTIME FUNCTION

```
def chkinfo():
    info = f"|Current set HR: {Main.set_hr}|Current Max HR {Main.max_hr}|Current Min HR {Main.min_hr}|Average Heart Rate {Main.avghr}|"
    return info
```

FIG 7.22 – CHKUSR FUNCTION

A login system has also been incorporated into this prototype system and users can be added and removed from the device via the `addusr` and `delusr` commands seen in fig 7.23. These two commands use an external csv file which stores the username and hashed password for added security seen in fig 7.24.

```
def delusr(option):
    found = False
    with open('userdata.csv', 'r') as ud:
        userdata = dict(reader(ud, 'udata'))
    for users in userdata:
        if option == users:
            os.remove('userdata.csv')
            found = True
            break;

    if found:
        del userdata[option]
        with open('userdata.csv', 'wb') as ud:
            for key in userdata.keys():
                ud.write((f'{key}|{userdata[key]}\n').encode('utf-8'))
            out = f"User {option} Deleted"
        SysAdmin.loaduserdata()
        return out
    if not found:
        out = f"User {option} Was not Found"
        return out

def addusr(uname, passwd):
    for key in SysAdmin.userdata.keys():
        if uname == key:
            out = f"User {uname} Already exists"
            return out

    hpasswd = (SHA256.new(passwd.encode('utf-8'))).digest()
    SysAdmin.userdata.update({uname:hpasswd})
    os.remove('userdata.csv')
    with open('userdata.csv', 'wb') as ud:
        for key in SysAdmin.userdata.keys():
            ud.write((f'{key}|{SysAdmin.userdata[key]}\n').encode('utf-8'))
        out = f"User {uname} Added"
    SysAdmin.loaduserdata()
    return out
```

FIG 7.23 – USER COMMANDS

The final command is incorporated to change the heart health variable seen within the base pacemaker program, the command takes a value from (0-4) this function can be seen in fig 7.25 and the information it changes can be seen in fig 7.3.

```
def hearthealth(health):
    if -1 < int(health) < 5:
        Heart.hearthealth = int(health)
        out = f"HH set to {health}"
    else:
        out = "ERROR"
    return out
```

FIG 7.25 – HEARTHEALTH FUNCTION

7.2.4 CONNECTIVITY

This section consists of two scripts: The Prototype pacemaker client and the prototype pacemaker server. These are a required feature of the program and allow for connections to be made to the implanted device via a Wi-Fi connection. This is made possible via the socket library as it allows ports to be opened with ease via a few lines of code which can be seen in fig 7.26. The Server and client use encrypted communication to ensure that any confidential information such as passwords and medical data cannot be intercepted in plain text; this encrypted communication is possible using the ECB class seen in the logging section above this can be seen in fig 7.27. When the client and server are using the same encryption key a connection will be made between the two prompting the client to login, the default username and password that can be deleted and replaced with a more secure user is admin:pacewall and can be seen in the hashed form in the fig 7.24. The login section can be seen in fig 7.28 and prevents unauthorized users from connecting to the system, in a later version the client may be made slightly more lightweight by relying more on the server for outputting all information. The login system also incorporates an alert if a username or password is entered that is incorrect. The Server will also disconnect the user if any malicious commands are detected, or any errors are detected which prevents the pacemaker from crashing. The server also incorporates the logging function and logs: user commands, command outputs, user connects and disconnects.

```
while True:
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.bind((host, port))
        s.listen()
        conn, addr = s.accept()
```

FIG 7.26 – SERVER INITIALIZATION

```
data = ECB.decrypt(conn.recv(1024).decode('utf-8'), epass)
conn.sendall((ECB.encrypt("INVALID COMMAND", epass)).encode('utf-8'))
```

FIG 7.27 – ENCRYPTED COMMUNICATION

The server is also capable of detecting when a command has been sent and stripping it to get the value the user is entering, this command detection is carried out by the switch function and works via detecting the command in the data that is sent from the client and example of this can be seen in the add user command being detected in 7.29. a command is not valid the server will output back to the client an invalid alert.

```

while login:

    try:
        data = ECB.decrypt(conn.recv(1024).decode('utf-8'), epass)
    except:
        s.close()
        connected = False
        print(f"{datetime.datetime.now().strftime('%Y-%m-%d %H:%i:%S')}: Lost Connection to {addr}")
        logtxt = f"{datetime.datetime.now().strftime('%Y-%m-%d %H:%i:%S')}: Lost Connection to {addr}"
        LC.log(logtxt)
        break

    if data in SysAdmin.userdata:
        uname = data
        conn.sendall((ECB.encrypt("True", epass)).encode('utf-8'))
        try:
            data = ECB.decrypt(conn.recv(1024).decode('utf-8'), epass)
        except:
            s.close()
            connected = False
            print(f"{datetime.datetime.now().strftime('%Y-%m-%d %H:%i:%S')}: Lost Connection to {addr}")
            logtxt = f"{datetime.datetime.now().strftime('%Y-%m-%d %H:%i:%S')}: Lost Connection to {addr}"
            LC.log(logtxt)
            break

        if str(f"{SHA256.new(data.encode('utf-8')).digest()}") == SysAdmin.userdata[uname]:
            conn.sendall((ECB.encrypt("True", epass)).encode('utf-8'))
            conn.sendall((f"Welcome To Pacewall: {uname}", epass)).encode('utf-8'))
            print(f"{datetime.datetime.now().strftime('%Y-%m-%d %H:%i:%S')}: {uname} has logged on")
            logtxt = f"{datetime.datetime.now().strftime('%Y-%m-%d %H:%i:%S')}: {uname} has logged on"
            LC.log(logtxt)
            login = False

        else:
            print(f"{datetime.datetime.now().strftime('%Y-%m-%d %H:%i:%S')}: !ALERT!{addr}->##### INVALID PASSWORD")
            logtxt = f"{datetime.datetime.now().strftime('%Y-%m-%d %H:%i:%S')}: !ALERT!{addr}->##### INVALID PASSWORD"
            LC.log(logtxt)
            try:
                conn.sendall((ECB.encrypt("False", epass)).encode('utf-8'))
            except:
                s.close()
                connected = False
                print(f"{datetime.datetime.now().strftime('%Y-%m-%d %H:%i:%S')}: Lost Connection to {addr}")
                logtxt = f"{datetime.datetime.now().strftime('%Y-%m-%d %H:%i:%S')}: Lost Connection to {addr}"
                LC.log(logtxt)
                break

    else:
        print(f"{datetime.datetime.now().strftime('%Y-%m-%d %H:%i:%S')}: !ALERT!{addr}->{data} INVALID USERNAME")
        logtxt = f"{datetime.datetime.now().strftime('%Y-%m-%d %H:%i:%S')}: !ALERT!{addr}->{data} INVALID USERNAME"
        LC.log(logtxt)
        try:
            conn.sendall((ECB.encrypt("False", epass)).encode('utf-8'))

```

FIG 7.28 – ENCRYPTED COMMUNICATION

```

if 'addusr' in data:
    sdata = data[7:].rsplit(sep=' ')
    result = SysAdmin.addusr(sdata[0], sdata[1])
    conn.sendall((ECB.encrypt(result, epass).encode('utf-8'))
    print(f"{datetime.datetime.now().strftime('%Y-%m-%d %H:%i:%S')}: {(len(uname) * ' ')}->{result}")
    LC.log(f"{datetime.datetime.now().strftime('%Y-%m-%d %H:%i:%S')}: {(len(uname) * ' ')}->{result}")
    return

```

FIG 7.29 – ENCRYPTED COMMUNICATION

7.3 Prototype Pacemaker

The Software was developed as multiple python files that all work together these files are: main.py, ppmClient.py, ppmServer.py, SysAdmin.py, logger.py the use to external files called userdata.csv and Log.txt. Both of these external files using encryption or hashing to ensure security of the stored content.

7.3.1 MAIN.PY

The prototype solution is designed for a pacemaker and for this reason a simple pacemaker has been designed inside main.py. This file also contains functions that simulate multiple types of heart beats: Healthy, Slight Issue, ill, death and sporadic. These types of heart beats allow for the program to be tested to its limits and shows how the intrusion systems will react. There are three functions that make up the pacemaker software: pm, which is the base function that class the other functions within the main class and essentially runs the logic on whether or not the pacemaker should sent a pulse to the heart; alert, which determines if the average heart beat is lower than the min heart rate set by the doctor; avgcal, that detects and calculates the avg heart beat over 5 heartbeats based off of the time between each heartbeat. The second class in main is in charge of simulating the heartbeat and generates a time until the next heartbeat by using the heart health settings discussed above.

7.3.2 SYSADMIN.PY

The program uses another python file called SysAdmin which allows for modification to be made to the variables within the main python file, it allows for the following commands to be used: setmaxhr, which sets the maximum heart rate before the system uses the alert function to log that the heart rate has exceeded the max heart rate; setminhr, sets the minimum heart rate before the system uses the alert function to log that the heart rate is below the safe level; setavghr, is uses the calculate what the average heart rate should be kept at and lets the pacemaker pulse if the heart rate is lower than this value; chkinfo, outputs the avg/min/max and detected heart rate; delusr & addusr, allows for users to be added and deleted from the system and uses hashed passwords; chkusr, outputs the users in the system; chkuptime, outputs the systems uptime; hearthealth, is only used in the prototype and allows for the hearts health to be changed.

The SysAdmin class is not able to interact with the main python file instead it uses an intrusion prevention system class called Pacewall which takes inspiration from another similar solution called CardiWall and is used for setting the maximum, minimum and average heart rate that can be set for the pacemaker.

7.3.3 LOGGER.PY

The logger file consists of the class LC with the function log, the class stores the current line number since start-up as sfcnt and the encryption password as epass. The log function then takes the data that is to be logged and encrypts it then outputs it to the log file.

7.3.4 PPMSERVER.PY

The prototype solution allows for external connection to be made to the pacemaker which will allow for sensitive variables to be edited using the SysAdmin file and the PaceWall class. The ppmServer is the file that is run on the pacemaker and is capable of starting up the pacemaker function, it is capable of calling all the functions within the SysAdmin class. The program also uses the Logger file to output all of its data to the Log.txt mentioned above. The server uses timestamps and line numbers for each start-up which allow for tampered logs to be seen. The server also requires the user to have the correct encryption password set on the ppmClient.py to create a connection as all communication is encrypted with AES and requires a username and password before variables can be edited. The username, command and output are all logged along with the IP that has connected and logged in as the user.

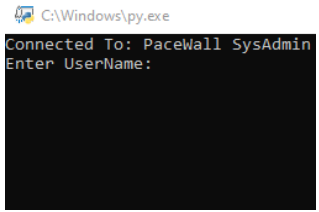

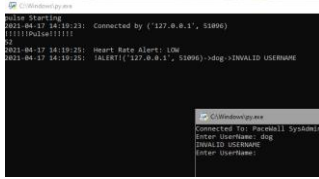
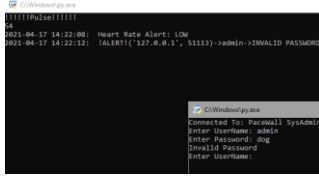
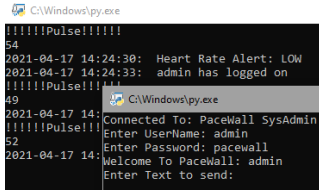
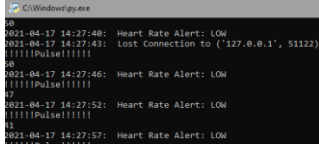
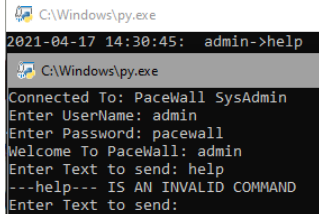
7.3.5 PPMCLIENT.PY

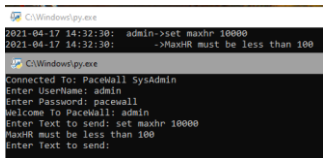
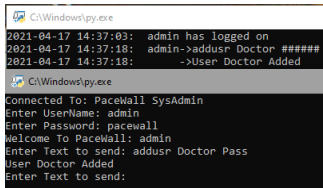
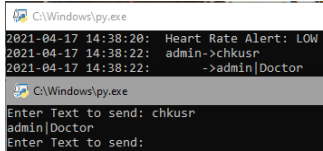
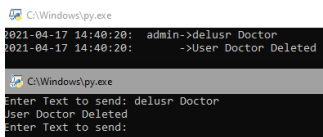
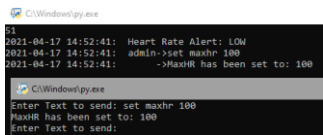
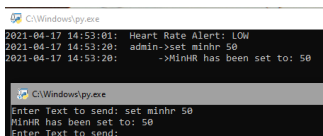
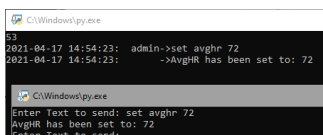
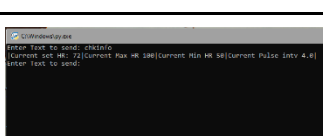
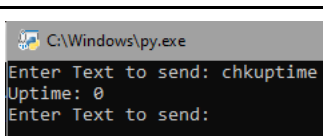
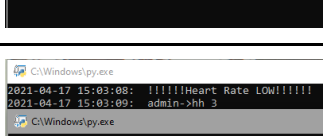
The Client used to connect to the pacemaker requires an encryption password to communicate with the server so that confidential data such as passwords cannot be intercepted by anyone sniffing the connection. The client does not use any logging as the server can log each client's activity. The client does not consist of much code and essentially allows for a user to login and send and receive data, it is very lightweight as this improves device safety and means that the client can connect to multiple patients' pacemakers.

8. ARTEFACT TESTING/EVAL

8.1 Testing

The following testing table is used to show how the program is functioning correctly and is working to meet the requirements set out in the Moscow diagram fig 3.1.

Test	Expected	Occurred	Passed	Image
Connection To Server	The client should connect to the server and be prompted to login.	The Client Connected to the server and was prompted to login	pass	
Server Logs Connection	The Server will log the connection and output the address of the connection	The server logged and outputted the address of the connection	pass	
Server Invalid Login UserName	Server should output and log invalid username and the address that tried to use it.	The Server output the username was invalid and what address attempted to login with it.	pass	
Server invalid Login Password	Server should output and log invalid password and the username that was used and the address that tried to use it.	The server outputted the address, username and invalid password	pass	
User Welcome / Connected	The server should log that a user has connected and set a welcome message to the client	The server logged that the user admin had connected and sent a welcome message to the client.	pass	
User Disconnected	The server should state that the client has disconnected and log the address.	The server logged that the address had disconnected.	pass	
User Invalid command	The server should state that the user has entered and invalid command	The Server stated the user used an invalid command	pass	

User Invalid command value	The server should alert the user that the value they entered for a command was invalid.	The server alerted the user that the commands value was invalid.	pass	
User addition	The server should add a user and hide the users password	The Server added the user and hid the users password	pass	
Check users	The server should outputs the users stored within the usrerdata.csv	The server displayed the current users for the system	pass	
User Removing	The Server should remove the user from the userdata.csv	The server removed the user from the userdata.csv	pass	
Set max Heart Rate	The server should change the max heart rate value	The Max Heart rate was set to the user state variable	pass	
Set Min Heart Rate	The server should change the min heart rate value	The Min Heart rate was set to the user state variable	pass	
Set Average Heart Rate	The server should change the average heart rate value	The Average Heart rate was set to the user state variable	pass	
check info	The server should output the min,max,avg and target heart rate.	The Server outputs the min,max,avg and target heart rate.	pass	
check up-time	The Server should output the up-time in days	The server outputted the up-time of zero days	pass	
heart health	The server should change the heart health value	The Heart health was changed to the users value	pass	
Heart LED	The Red LED should flash every time there is a heart beat	The Led Flashed in sync with the heart beat	pass	

Pulse LED	The Blue LED should flash if the pacemaker has had to pulse.	The Led Flashed in sync with the pulse	pass	
PaceMaker Heart LED	The Green LED should flash if the heart LED flashes or the Pulse LED flashes.	The Led Flashed in sync with the heart beat and pulse	pass	

8.2 Evaluation

The software was given set requirements in the beginning in this project, these were stated in the fig 3.1. To ensure that the project has been successful a review and evaluation of how many of these elements have been achieved is necessary. The software was required to be capable of base pacemaker functionality, this means that it must be able to detect the heart rate and output a pulse to alter it if it deviated from the target heart rate, this has clearly been achieved and can be seen in fig 7.8, fig 7.3 and fig 7.4. The program was required to use three key variables the min, max and target heart rate, these have also been added and can be seen being modified in the table above. Integration with LEDs was another feature that has been implemented and once again can be seen in the table above and in fig 7.5. The Software was also required to be able to be integrated on a microprocessor such as a Raspberry pi. This integration can be seen in the last three sections of the table above. The Application also needed to incorporate wireless connectivity the chosen method for this project was wireless due to ease of use and integration, this can be seen in the connectivity section above. The final necessary feature was the Intrusion detection system which took the form of a cryptographic log, this can be seen in fig 7.11. The program was also able to incorporate encrypted communication via the same system used for the cryptographic log, and due to the artificial heart class is capable of running in a sandbox state. The program also incorporated a client which acted as an external application to connect to the prototype device.

The final two sections of the Moscow diagram: could have and should have, prompts five features with one of these being implemented into the prototype system for extra security. This feature being the implementation of a similar system called cardi-wall which is mentioned in the intrusion detection systems literature review. The feature allows for the prevention of malicious variable changes. The program also implemented another layer of security by developing a login system that was not established as a feature in the Moscow diagram but seemed necessary for client-server connection.

Overall, the prototype for a pacemaker intrusion detection system has well exceed requirement expectations and even implemented features that were not originally anticipated proving that this project has been successful and achieved an ample solution for securing pacemakers that opt to become an IOT device.

9. CONCLUSION

This research paper set out to determine ample solutions for developing a software prototype for a pacemaker intrusion detection system. The Paper discusses how the rapid advancements of IOT has led to medical devices being modified to be included in this bracket poses potential security concerns a current solution used by St. Jude Medical is also mentioned which poses some of these concerns due to allegations made against the company. From the security concerns brought up from this issue a set of functional requirements were drafted up to show what a secure prototype pacemaker system should incorporate; these requirements can be seen within the functional requirements section in the form of a Moscow diagram.

To assist in developing a solution and writing a paper to discuss the findings around this topic a Project Gantt chart was constructed, this gnat chart allowed for key elements to be baked into the project such as midpoint review and the deadline for the paper. To accomplish the goal of developing a software prototype for a pacemaker intrusion detection system the paper reviews the following topics: Pacemakers, which investigated the vulnerabilities, history and types of current pacemakers; attack mitigation, which reviewed current and novice solutions for mitigating against an attack; Microprocessors, which reviews current hardware systems that could be used to develop the prototype solution on; connectivity, shows the types of connectivity that could be used with the system; Encryption, provides multiple types of solutions to encrypting the cryptographic log as well as communication; Language, provides an investigation on which language would be best suited to this project.

After this research was carried out the development of the prototype pacemaker took place, the pacemaker was designed using the recommendations state within the literature review to best suit the problem at hand as well as meet the projects' function requirements stated in the beginning of the paper, this was carried out by developing 5 python files: ppmClient, ppmServer, SysAdmin, main and Logger.

The project in its whole originally started out to develop a solution to this lack of security and through both the literature review and project development this goal has been achieved, this can be seen by the testing and evaluation section above which shows how the program provides ample security and shows that a software prototype for a pacemaker intrusion detection system has been developed. Due to this being a prototype solution professional implementation of this system should undergo extended research and improvements stated within the Moscow diagram should be implemented, the hardware used for the prototype should also be built upon to minimise the size and create a medically safe device for implanting within the human chest cavity.

REFERENCES

- O'Reilly Media, Inc., 2016. *Arduino: A Technical Reference* by J. M. Hughes. [Online]
Available at: <https://www.oreilly.com/library/view/arduino-a-technical/9781491934319/ch01.html#:~:text=In%202005%2C%20building%20upon%20the,Institute%20lvrea%20in%20lvrea%2C%20Italy.>
[Accessed 18 May 2021].
- Abbott, 2021. *MERLIN@HOME™ TRANSMITTER*. [Online]
Available at: <https://www.cardiovascular.abbott/us/en/hcp/products/cardiac-rhythm-management/merlin-home-transmitter.html>
[Accessed 18 April 2021].
- Arduino, 2018. *What is Arduino?*. [Online]
Available at: <https://www.arduino.cc/en/Guide/Introduction>
[Accessed 18 May 2021].
- Arduino, 2021. *Store*. [Online]
Available at: <https://store.arduino.cc/>
[Accessed 18 May 2021].
- Awan, M. F. et al., 2018. *Experimental phantom-based evaluation of Physical Layer Security for Future Leadless Cardiac Pacemaker*. Bologna, Italy, IEEE.
- Barrett, S. F., 2013. *Arduino Microcontroller Processing for Everyone!*. 3rd ed. s.l.:Morgan & Claypool.
- Brook, E. W., 2020. *Ensuring Security of Human-Computer Integrated Devices in Insecure Environments*, Stoke-on-Trent: Staffordshire University.
- C. Walton Lillehei, M. P., Vincent L. Gott, M. & Paul C. Hodges Jr., M., 1960.
TRANSISTORPACEMAKERFORTREATMENTOFCOMPLETEATRIOVENTRICULARDISSOCIATIO. *JAMA: The Journal of the American Medical Association*, 172(18), pp. 2006-2010.
- Davis, C., Muthineni, A. & John, E., 2019. *Low-Power Advanced Encryption Standard for Implantable Cardiac Devices*. Dallas, TX, USA, IEEE.
- Fu, K. & Blum, J., 2014. Controlling for Cybersecurity Risks of Medical Device Software. *Biomedical Instrumentation & Technology*, Volume 48, pp. 38-41.
- Gast, M. S., 2005. *802.11 Wireless Networks: The Definitive Guide*. 2nd ed. Sebastopol, California: O' Reilly Media, Inc..
- Geselowitz, M. N. & Leder, R. S., 2017. *Pacemakers: Engineering and Technology History Wiki*. [Online]
Available at: <https://ethw.org/Pacemakers>
[Accessed 18 May 2021].
- Greatbatch, W., 1962. *Cartridge piercing mechanism or the like*. Unite States of America, Patent No. 3,051,356.
- Haartsen, J., 2000. The Bluetooth radio system. *IEEE Personal Communications*, 7(1), pp. 28-36.
- Heath, N., 2018. *Inside the Raspberry Pi: The story of the \$35 computer that changed the world*, s.l.: TechRepublic.

- Hellman, M. E., Diffie, B. W. & Merkle, R. C., 1977. *Cryptographic apparatus and method*. United States of America, Patent No. 4,200,770.
- Jiang, Y., 2017. *Program Analysis Based Bloatware Mitigation and Software Customization*, Pennsylvania: The Pennsylvania State University.
- Joan, B., 2010. *Difference Between AES and Twofish*. [Online]
Available at: <https://www.whysisdifference.com/technology/difference-between-aes-and-twofish.html>
[Accessed 18 April 2021].
- Kintzlinger, M. et al., 2020. CardiWall: A Trusted Firewall for the Detection of Malicious Clinical Programming of Cardiac Implantable Electronic Devices. *IEEE Access*, Volume 8, pp. 48123-48140.
- Kravitz, D. W., 1991. *Digital signature algorithm*. United States of America, Patent No. 5,231,668.
- Morrison, B. J., 2008. Putting the learning curve in context. *Journal of Business Research*, 61(11), pp. 1182-1190.
- National Institute of Standards and Technology, 2001. *Federal Information Processing Standards Publication 197*. [Online]
Available at: <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf>
[Accessed 18 April 2021].
- Premeaux, E. & Evans, B., 2011. *Arduino Projects to Save the World*. 1 ed. s.l.:Apress.
- Rapid Electronics Limited, 2021. *rapidonline*. [Online]
Available at: <https://www.rapidonline.com/>
[Accessed 18 May 2021].
- Raspberry Pi, 2021. *Products*. [Online]
Available at: <https://www.raspberrypi.org/products/>
[Accessed 18 April 2021].
- Raspberry Pi, 2021. *Raspberry Pi 4 Tech Specs*. [Online]
Available at: <https://www.raspberrypi.org/products/raspberry-pi-4-model-b/specifications/>
[Accessed 18 May 2021].
- Raspberry Pi, 2021. *SD Cards*. [Online]
Available at: <https://www.raspberrypi.org/documentation/installation/sd-cards.md>
[Accessed 18 April 2021].
- Schneier, B. et al., 2002. On the Twofish Key Schedule. *Selected Areas in Cryptography*, Volume 1556, pp. 27-42.
- Shamasastri, R., 1956. *Kautilya Arthashastra*. 2019 ed. s.l.:Parimal Publication Pvt. Ltd..
- SocialCompare, 2021. *Raspberry Pi 3*. [Online]
Available at: <https://socialcompare.com/en/review/raspberry-pi-3>
[Accessed 18 April 2021].
- Thamilarasu, G., Odesile, A. & Hoang, A., 2020. An Intrusion Detection System for Internet of Medical Things. *IEEE Access*, Volume 8, pp. 181560-181576.
- Thamilarasu, G., Odesile, A. & Hoang, A., 2020. An Intrusion Detection System for Internet of Medical Things. *IEEE Access*, Volume 8, pp. 181560 - 181576.

University of the Basque Country, 2015. *Pacemakers with Internet connection*. [Online]
Available at: <https://healthcare-in-europe.com/en/news/pacemakers-with-internet-connection.html>
[Accessed 18 April 2021].

Vallat, R., 2018. Pingouin: statistics in Python. *Journal of Open Source Software*, 3(31), p. 1021.

APPENDIX

All code for this project is available at the following GitHub repository: <https://github.com/Ebroox10/Final-Year-Project-Protoype-Pace-Maker.git>