A Security Analysis of Internet-connected Blood Pressure Monitors

Timothy Murphy

4th Year Project Report
Computer Science
School of Informatics
University of Edinburgh

2021

Abstract

Wearable, internet-connected, medical monitoring devices are becoming more popular, allowing users to frequently monitor their health without having to travel to a hospital. As the nature of the information being uploaded to the internet is of a sensitive nature, users would hope that it was being handled in a secure manner. But, with the rapid growth of the wearable healthcare market, device security has fallen by the wayside. In this study, I conducted an in-depth security analysis of three blood pressure monitors, uncovering cases of illegal user location monitoring, insecure Bluetooth pairing and the potential for thousands of users' personal and medical information to be exposed. Attackers could locate where a user lives, alter medication dosages or create fake blood pressure readings, leading to potential harm and distress for the user. To remedy the issues that I uncovered, I provide a list of recommendations for the vendors to improve their security and remove the vulnerabilities.

Acknowledgements

I would like to express my greatest thanks to my supervisor, Dr. Paul Patras, who guided me through this project and showed me how fascinating the world of computer security really is.

I also want to thank my family, who supported me through an exceptionally tough year and were always there for me when I needed them.

Lastly, I want to thank my friends in Edinburgh, and back at home, who would listen and talk to me during the highs and lows of my time at University.

Table of Contents

Chapter 1

Introduction

The Internet of Things (IoT) market has seen significant growth in the last decade, with the wearables market growing 30% every year and there being an estimated 601 million wearable devices in 2020 [26]. One third of these devices belong to a group known as the 'Internet of Medical Things' (IoMT), IoT devices specifically designed for healthcare settings. The IoMT in particular is expected to be worth $135 billion by the end of 2025 [13] and currently 60% of healthcare organisations make use of these devices [10]. There is a need to ensure these devices and the data they produce are kept secure and private.

The IoMT will be an incredible help to the the medical industry as it will help to lower costs and stress on healthcare systems. Internet-connected devices such as BPMs, ECG-capable wristbands, etc. allow patients to be monitored, even after they are discharged from hospital [6]. This is particularly helpful during the pandemic, as it allows monitoring of patients without them having to come into hospital, lowering the risk of them contracting or spreading Covid-19. As the average age of the population increases and an increased number of older people require medical care, these devices will help to reduce the burden placed on the healthcare industry [29]. Patients will be more comfortable with these devices rather than traditional equipment as they can be small form and measure continuously without disturbing them [29]. As patients can wear these devices continuously, and many more measurements can be taken, symptoms of diseases can be detected much earlier than they normally would, increasing the likelihood of successfully treating the patients before the disease becomes more serious. Blood pressure monitors in particular can be used to detect high blood pressure, commonly caused by hypertensions, leading to illnesses which kill around 7.5 million people every year [29]. The NHS was understaffed by around 100,000 people in 2018 [50], implementing these devices could help to reduce that number. With these devices being fitted once and then taking many measurement, the chances of human errors affecting the measurements is much less likely [6].

People who live far from a hospital, lack public transportation or those who do not have access to healthcare would all benefit from these monitoring devices, as they will give them access to some form of health monitoring. Healthcare staff will have more time to tend to patients who require more personal care as they will save on the time

they would normally use to take patients measurements, such as blood pressure. In the case of an accident, the patients can be fitted to some IoMT devices to begin taking measurements and then send them to the hospital for immediate analysis without any delay. These devices also reduce the amount of resources required for manual data entry as this can all be done automatically [52].

While the IoMT improves patient care, it also helps with the day to day running of hospitals and other healthcare areas. 80% of healthcare executives believe that the IoMT has helped innovation and 73% said they have experienced cost savings after adopting IoMT devices [8].

As people rely on many of these devices to alert them of any decreases in their health, it is vital that the information they measure and transfer is kept secure and its integrity is maintained. Without security and integrity being at the forefront of the of these devices, users' confidential medical information could be at risk of being exposed, as well as altered to create panic and distress for the user. If readings that show a deterioration in the user's health are obscured then illnesses that could be treated early on may not be detected for months, potentially leading to serious health complications.

1.1 Project Overview

In my project I conducted an in-depth security analysis of three representative BPMs and the entry ecosystem that governs their operation (i.e. interfacing with smartphones, communications with the cloud, etc.). These devices are the MocaCare MocaCuff, the Omron Evolv and the Activ8rlives blood pressure. The MocaCuff is a wrist blood pressure monitor while the other two attach to the upper arm. All three devices connect to a smartphone application which they use to transmit data to, which is then uploaded to a server particular to each device. Due to the sensitive nature of the data being transmitted, one would hope that this would be handled in a secure manner.

My goals in this projects were to thoroughly investigate different aspects of these blood pressure monitors and their communication models. I created a test bed which allowed me to SSL strip any traffic between the phone app and web server, as well as used Bluetooth debugging tools to monitor the pairing process and data advertisement of the blood pressure monitors themselves. The test bed is further detailed in section 4.2.

Regarding the pairing process between phone and monitor, I found that the MocaCuff did not bond with the devices it paired with - this meant that the monitor could be paired with by any device every time it was switched on, allowing any device that was searching for a MocaCuff to download all previous blood pressure readings of the user.

In my initial testing of the communication between the phone app and server, I discovered that the MocaCare Android app was sending location information, bundled inside blood pressure readings, to the server without the user's knowledge or consent. This was a clear violation of a user's privacy and the GDPR.

This testing also revealed that server access tokens had no expiry on either the Activ8 or MocaCuff blood pressure monitor. The implications of this could be severe if an attacker were to gain knowledge of the access token - this would allow them to read

and write to a user's account indefinitely, only losing access once a user changed their password. Some of the information the attacker could read and write to include the user's name, address, blood pressure readings and medication reminders. All of this is detailed further in Chapter 5.

After experimenting with the data that would be sent to and from the MocaCare servers, I discovered a flaw in the web API that allowed a user to download blood pressure readings for an account by only sending a locally generated hash of the account ID. The account IDs were consecutive (i.e. the 7000th account would have the account ID 7000), so by extracting the hashing algorithm **I was able to hash all account IDs and download the blood pressure readings of the entire userbase.** An attacker aware of this flaw would not only be able to download a user's blood pressure readings but, because of the **undisclosed collection of location information,** this sensitive data would also be able to do be downloaded, regardless of where the attacker was located.

Another serious security flaw in the MocaCare web API was a way to access the **names, usernames, email addresses and profile pictures of the entire userbase by exploiting part of the built in messaging service in the MocaCare app.** An attacker with knowledge of the flaw, in conjunction with the ability to obtain blood pressure readings, could build profiles of the users of this service, knowing their name, email address, home address, blood pressure readings and a photo of them.

While looking into the websites for the various services, I discovered that the MocaCare website login and account service did not implement HTTPS and all login information and account information was sent in clear text, including blood pressure readings and account passwords.

Once I understood the communication model and web API used by the different devices and services, I moved onto attempting to find vulnerabilities that would allow me to modify and insert my own data into a user's account. By using a man-in-the-middle attack I was able to alter data going to and from the server and phone on the Activ8 and Moca apps. Also, with knowledge of the access token, an attacker could craft their own Rest API requests that would allow them to delete, modify and create their own data on a user's account for all three services.

Based on the vulnerabilities discovered, I made a set of recommendations to remedy these, as well as to guide the development of future products developed by other vendors.

1.2 Responsible disclosure

The nature of my findings in this project are of a sensitive nature and could leave users of these 3 services vulnerable to the attacks I discovered. Therefore, I am actively working with my supervisor to prepare a summary report of the findings, in view of disclosing them to the vendors.

1.3 Paper publication

I am in the process of writing a paper based on my findings with the MocaCare app and monitor which I hope to be published in the coming months. The paper will be submitted to a specialised IoT security workshop co-located with one of the major ACM/IEEE conferences.

Chapter 2

Background

The IoMT encompasses a vast number of different devices, including blood pressure monitors, sleep monitors and facial disorder detection [5]. Due to their benefits there has been a rush to introduce IoMT devices into healthcare settings, but this has introduced a few problems. As there are many different models and manufacturers of these devices, compatibility issues are arising between these devices as there is a lack of standardisation [24]. Also, while these have grown in popularity, their security has not been in the forefront of the healthcare industry, leading to many issues of patient confidentiality and the handling of sensitive information.

A number of these devices also require an application, such as one on a smartphone, to control them and receive the measurements they take. These applications, when not developed with security in mind, could be potential risks to patients confidentiality if a security flaw is found within them. If the application is freely available for people to download then it leaves them open to being reverse engineered to make it easier to find these security flaws.

There have already been a number of security issues with the adoption of the IoMT, with 89% of healthcare organisations saying that they have experienced security breaches after adopting these devices and 49% claiming that they have had malware issues. In 2017, 45% of all ransomware attacks were in healthcare settings and it was recently shown that healthcare experiences double the amount of attacks that any other industry does [7].

Flaws in these devices continue to be discovered, with a recent study showing that there were over 8000 exploits in pacemakers from four different manufacturers [11]. A recent survey showed that only 17% of device manufacturers were taking steps to prevent attacks on devices that could have adverse affects on patients, despite 67% of device makers believing an attack on their devices would be likely within 12 months [47]. Only 9% of device manufacturers test their devices every year and 43% do not perform security tests at all [47]. A Hewlett-Packard study, conducted in 2014, showed that there was no support for strong encryption nor secure programming on 70% of IoT devices, IoMT included [41].

The blood pressure monitors that I will be testing are not medical grade which would

require the devices to have a certain degree of accuracy, and therefore do not fall under the IoMT umbrella and are targeted more towards home users. Regardless, the above concerns still show that more research and strong security requirements are needed to fully secure these devices, and lessons learned with the monitors I am testing could be applied to devices that are categorised as IoMT. Also, if the devices I am investigating can have their security strengthened then they could qualify as IoMT devices.

2.1 Regulations and Standards

There does not exist any laws or regulations with regards to the IoT, but there are manuals and guidance on how to keep them secure - some are detailed below.

For any program Confidentiality, Integrity and Availability (the CIA Triad) should always be kept in mind:

- Confidentiality, where data is only disclosed to those authorised to view it.
- Integrity, where data is not altered or destroyed unless done by those authorised to do so.
- Availability, where the program is accessible. These are key to creating a secure program, but are not always followed.

To ensure devices are as secure as they claim, the manufacturer can use the Common Criteria Evaluation Methodology, a framework where the manufacturer specifies their security functionality and a laboratory will test the device to evaluate these claims [1]. The methodology consists of a Protection Profile, which is a document specifying security requirements for a specific type of device - manufacturers can elect to have their device tested against one or many protection profiles. The manufacturer can produce a Security Target document, detailing the security properties of a device and how it complies with a Protection Profile.

There has recently been released a support program by the US Nation Institute of Standards and Technology to help with the development and application of standards and guidelines related to cybersecurity in the IoT [9]. They list various security concerns with the design of IoT devices and run workshops on security to help manufacturers of IoT devices to better understand how to make their devices secure. In December 2020, the NIST released 4 new documents to give manufacturers a starting point on what customers would expect when it comes to security. [2]

The IoT Security Policy Platform is a body of governmental organisations and industry organisations who work to ensure IoT devices are secure. The aim to promote good security practices in IoT devices to consumers, manufacturers, policy makers, regulators and retailers [45].

The European Telecommunications Standards Institute released a document containing standards to help make IoT devices secure. It is a culmination of good practices in security which should be used to help guide manufacturers of devices. The guidance is outcome focused, such as ‘Keep software updated’, allowing manufactures to adapt

the guidance to their particular product. As the document is just guidance, it is not necessary for manufacturers to follow it [17].

The European Union Agency for Cybersecurity released a study detailing good practice and guidance in IoT development, with a particular focus on software development. The study explored the cybersecurity challenges in IoT device development, key areas that needed to be protected on devices, explored different attack scenarios targeting IoT devices, developed security measures to help prevent against these attacks and released these measures as guidance to IoT device developers and manufacturers [18].

The Open Web Application Security Project - Internet of Things Project was started in 2014 to guide developers and manufacturers of IoT devices to make better decisions and in 2018 the OWASP released an IoT top 10 things to avoid when building IoT devices. With guidance such as 'Weak, Guessable or Hardcoded passwords', the advice is meant to be simple and be applicable to consumers, manufactures and enterprises [37].

A paper from RIPE NCC that was recently published details guidelines on security considerations for the use of IoT devices [42].

From all of the above, it can be seen that there has been a lot of push to research and create guidance around securing the IoT. But, without any regulation, manufacturers and developers are free to decide whether they follow any of the advice, allowing them to disregard the security of their devices when they release them to the public. For example, only 51% of device manufacturers follow FDA guidance to reduce security risks on the medical devices that they manufacture [47].

2.2 IoT Prior Research

There has been little research into the security of internet connected blood pressure monitors and IoMT in general, but research methods used in analysing the security of other IoT devices will be useful in my project.

[15] looked at the Fitbit eco system. They were able to find privacy leakages from intercepting unencrypted bluetooth and performing man-in-the-middle attacks to view data being transmitted over HTTPS. They were also able to install manipulated malware on fitbit trackers remotely, over bluetooth.

[19] also looked at Fitbit devices, specifically the Fitbit One and Fitbit Flex. They were able to reverse engineer the communication protocol between the phone and the server, allowing them to read personal information and inject their own false information.

[30] studied the Belking WeMo Home EcoSystem to try and find any security vulnerabilities. They were able to reverse engineer the Android smartphone app and find an exploit to read the password to the home WiFi. They were also able to imitate a WeMo device and have a phone connect to it, allowing a phishing attack to be launched against the user potentially leading the sensitive information being revealed.

[20] looked at 17 different fitness trackers to cover a variety of different manufacturers and feature sets. For all cloud-based devices, they were able to inject fake data despite

them transmitting over HTTPS. Most devices were not able to guarantee data integrity when in transit and none of the devices implemented end-to-end encryption.

[44] was able to demonstrate a potential attack on enterprises using a compromised smartwatch. The device was able to scan a network to map out the printers in an office and then imitate a printer service so that documents are sent to the device. All of this could be done without the smartwatch user being aware.

2.3 Testing methods in previous research

The monitors I will be looking at connect to a smartphone via Bluetooth 4.0 to exchange information, then that smartphone sends the data to a server via a wireless network, this is the same system that is used by the devices in the previously mentioned studies. Some of the testing methods are detailed below.

2.3.1 MITM attack

Many studies used a Linux based laptop as a wireless gateway to connect a phone to that has a faked CA certificate installed on it. This allows for the simulation of a man-in-the-middle attack, where all traffic between the phone and the server are monitored via SSL stripping on a device between them, as long as the applications do not implement certificate pinning. Using this method will allow me to analyse the communication protocol that the blood pressure monitors use. If I am able to reverse engineer the protocol then it may be possible to remotely transmit fake data to the servers, without needing to be in possession of a blood pressure monitor.

2.3.2 Android app

The Android app handles the data from the blood pressure monitor and forwards it onto the server, potentially with other data that is bundled in. If the app can be decompiled then it could be possible to find security flaws within the app which could then be exploited. [15] went further and developed their own version of the smartphone app to easily exploit the vulnerabilities in the app.

2.3.3 Hardware

If the communication between the device and the server is encrypted, then accessing the hardware could be another way of manipulating data that is sent to the server. This would involve tearing down the device and reverse engineering the hardware layout.

Chapter 3

Threat Modelling

A threat model is a process of identifying security flaws in a program or system. When created during the early stages of development of a project, the findings from the process can help to inform design, mitigating threats before they reach the user. The outcome of a threat model is to maintain the integrity of the CIA triad [51].

The OWASP details 4 questions that should be asked in a threat model: What are we building? What can go wrong? What are we going to do about that? Did we go a good enough job? [3]

All of the above questions would be suited for a business developing a computer system, but for my project I will be mainly focusing on 'What can go wrong?' as I search for threats in the system. I will also briefly cover 'What are we going to do about that?' by providing some recommendations for remedying the discovered issues.

To figure out what can go wrong, I need to identify potential threats that could occur in the blood pressure monitors, the phone application, the server and the link between them. To do this, I need to make some assumptions on what an attacker would be capable of and then hypothesise what the attackers goals may be.

3.1 Attacker Capabilities

For most attacks, I assume the attacker is in close proximity to the victim. They would be in Bluetooth range to their blood pressure monitor, or on the same WiFi network as the victim. The attacker would not need to have access to their own blood pressure monitor and would only require a computer and a smartphone. The vulnerabilities discovered in Chapter 7 would not require the attacker to be anywhere near the victim, and the attacks could be launched from anywhere in the world, only requiring an internet connection.

3.2 Attack Scenarios

I considered 3 main goals for an attacker: spying on the victim's personal and medical data, which is discussed in Chapters 5 and 7; altering a victim's data, discussed in Chapter 6; and launching a denial-of-service attack against the victim, which is discussed in both Chapters 5 and 6.

Monitors – Phone
Data that is transmitted between the blood pressure monitor and the phone they connect to could be read by an attacker via sniffing Bluetooth packets that are transmitted between the two devices. This could be done without the user being aware that the data is being read. Another method is for the attacker to use their own phone to connect to the blood pressure monitor and receive its readings - this would be dependent on the pairing process the devices use. Finally, an attacker could spoof a blood pressure monitor and advertise itself to the phone to receive data that is sent from the phone to monitor. This method could also allow the attacker to transmit fake readings to the phone.

Phone – Server
The phones likely send data to the web servers via HTTPS. If certificate pinning is not implemented , an attacker could launch a man-in-the-middle attack on the user's phone, allowing them to view all requests sent between them. Access to this traffic would allow the attacker to potentially view login information, personal user information, modify sent data and even craft their own requests to be sent to either phone or server. This attack method could be used for spying, modification of data and denial-of-service.

Server
An attacker could look into the Web API that is used to communicate between phone and server to try and discover any vulnerabilities that exist. This could range from denial-of-service, to compromising login information and personal information of accounts on the service.

Chapter 4

Methodology

4.1 Devices

Figure 4.1: MocaCare MocaCuff

Figure 4.2: Omron Evolv

Figure 4.3: Activ8rlives blood pressure monitor

4.1.1 MocaCare MocaCuff

MocaCare are a relatively small business, selling only their MocaCuff blood pressure monitor (Figure 4.1) and a wireless heart rate monitor, MocaHeart. The MocaCuff is a small BPM that attaches at the wrist. The MocaCuff won a CES Best Tech award [34]. The device connects to smartphones via a mobile application that is available on iOS and Android. The app has thousands of downloads on the Google Play store [35].

The mobile application is capable of storing readings from both Moca devices, displaying trends of previous blood pressure readings and adding notes to specific readings. The app also has other features such as reminders for taking medications, making doctors appointments and messaging with other users of the app. To use the app a user must create an account with their email address and set a password. Their profile can contain other personal information, such as weight, height, address and a profile picture.

With there being no prior studies on this device when I began, it was initially unclear how secure this device was.

4.1.2 Omron Evolv

Omron have a range of at home medical devices. The Omron Evolv (Figure 4.2) was in Forbes top 7 blood pressure monitors list for this year [28]. This device also has an Android and iOS app. The app has over a million downloads on the Google Play store [36].

The application has less features than the Moca app, with the device only recording blood pressure readings and allowing the user to add notes to them. The application can also connect with other fitness services, such as Google Fit.

There are prior studies into the accuracy of this device [49], but none into its security.

4.1.3 Activ8rlives Blood pressure monitor

Activ8rlives also sells many health and well being products. This blood pressure monitor (Figure 4.3) is the oldest of the three devices I looked at. The app has just over a thousand downloads on the Google Play store [31].

Just like the other two devices, this also has an Android and iOS app. The app has a range of features from a food diary to medication tracking.

This blood pressure monitor has no prior studies into its security.

4.2 Test Bed

To monitor the data that is sent from the blood pressure monitor to the phone, and then further onto the servers, I created a test bed, which is shown in Figure 4.4. The test bed consisted of a WiFi adapter that was connected to a computer running Ubuntu 20.04 [32]. A DHCP server was running on the computer, allowing a controlled network to be created.

A phone running Android 6.0.1, which the blood pressure monitors would connect to, was then connected to the controlled network. To intercept the data that the phone was sending onto the internet I set up a mitmproxy server on the computer [39]. mitmproxy allows me to SSL strip HTTPS traffic so I can lanuch a man-in-the-middle attack and see the data being transmitted onto the servers. To enable this to work, I also installed mitmproxy CA certificates onto the Android device that had the apps installed. This set-up allowed me to discover the vulnerabilities in sections 5.3 to 5.7, as well as launch the attacks in chapter 6.

4.2.1 Bluetooth

To debug the bluetooth communication between the blood pressure monitors and the Android phone, I installed Noridc nRF connect [12] on a separate Android device.

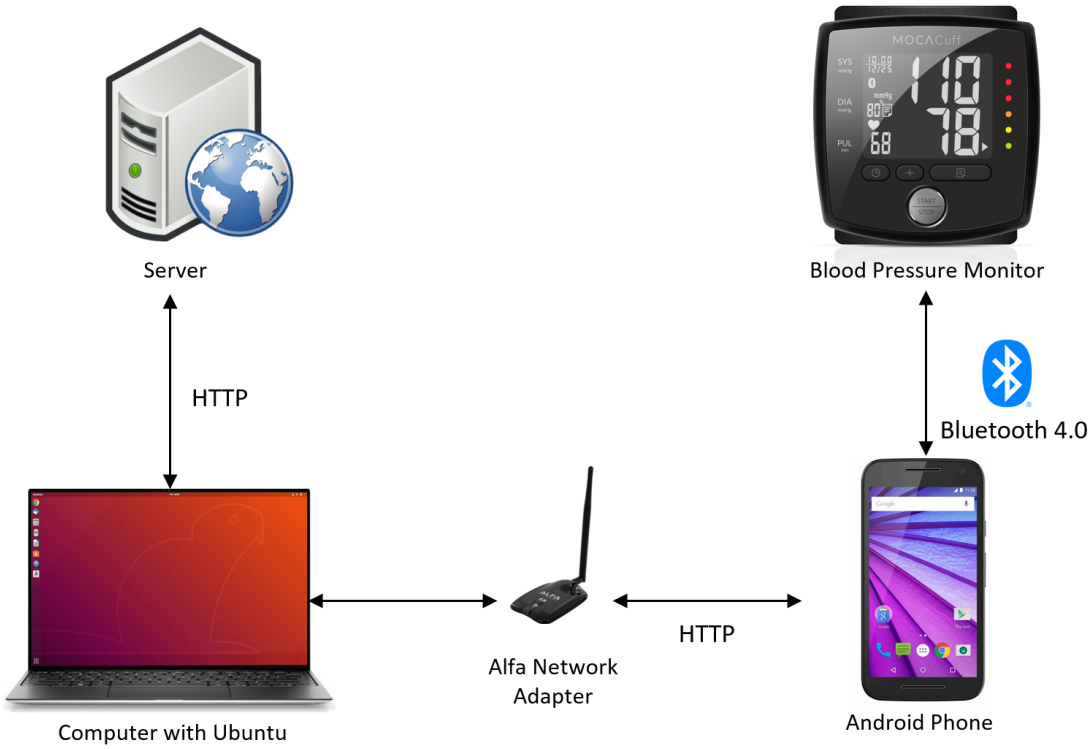

Figure 4.4: Test Bed

This allowed me to monitor the bluetooth devices in the area and attempt to connect to them to see what readable and writable information they were broadcasting.

4.2.2 Android Apps

To use the Android apps of the respective blood pressure monitors, I needed an email address to create an account. So, I created Protonmail accounts [4] which I then used to create accounts within the Android apps.

When I attempted to use mitmproxy to monitor traffic being sent to and from the Activ8rlives Android app no requests were being displayed. After some investigation I realised the app was bypassing the Android proxy settings. To force the traffic from the app through the mitmproxy server I install a VPN app called Drony [43].

In Chapter 7 I detail how I had to decompile one of the Android apps to try and find a hashing algorithm. Firstly, to obtain the APK file of the app I installed APK Extractor [48] on the Android phone. This created a separate APK file in an easily accessible directory of the phone.

To decompile the APK, I used Visual Studio code [33] with the APKLab plugin installed [46]. This set up would allow me to decompile any APK and look at its Java source code. I could then alter the code and recompile it, and use the android debug bridge [22] to install the altered APK to the phone. I could also extract Java functions and classes from the source code, allowing me to create my own Java programs.

To write and run Java code I used IntelliJ [14] and to send HTTPS requests to servers

en masse, and parse responses, I used Python [21] with the PycURL library [40].

4.3 Approach

Before beginning experimenting with the devices, I needed to plan my approach to how I would be testing. I based my testing plan off of the threat modelling research I did in Chapter 3.

- Sync with blood pressure monitor without authenticating **(Section 5.2)**
- Intercept traffic between phone and server **(Sections 5.3 to 5.7)**
- Manipulate traffic between phone and server **(Section 6.1)**
- Create fake messages and send to phone and server **(Section 6.2)**
- Flood app with fake data **(Section 6.2)**
- Investigate server end security **(Chapters 5 and 7)**

Intercepting traffic between the phone and server will be my first goal, as this will allow me to understand how the two devices communicate with one another, this is discussed in Chapter 5. With this knowledge, I will be able to manipulate traffic and create my own requests to send to the server in Chapter 6.

Chapter 5

Security Analysis

The data a blood pressure monitor stores and transmits is of a very sensitive nature. In addition, the apps these devices connect to often ask the user for other personal and medical information, including health conditions. These apps and servers also record addresses, phone numbers and email addresses. A user would expect this data to be handled in a very secure manner, with their privacy always being respected.

In this chapter, I firstly explain the communication model that the devices implement and detail the technologies and protocols that they use. I then discuss an in-depth security analysis I conducted on how data is handled on the blood pressure monitors, on the phone and how the data is transmitted between the monitor, phone and server. I started by looking into the pairing process between phone and monitor in Section 5.2 and then move onto understanding the messaging sequence between app and server. After gaining this understanding I was then able to validate my hypothesises of avenues an attacker may take to expose this data, and my findings in this chapter allow me to then craft my own REST API requests in Chapter 6.

5.1 Communication Model

All three blood pressure monitors use Bluetooth 4.0 to connect to the user's phone and share information. The phones communicate with the server using REST API requests via HTTPS. To be able to efficiently find vulnerabilities with these devices, I need to understand the communication model of these services. In the following sections I briefly breakdown Bluetooth 4.0 and HTTP so I can then use this information to thoroughly analyse the these devices.

5.1.1 Bluetooth 4.0

Bluetooth 4.0, more commonly known as Bluetooth Low Energy (BLE), is the wireless technology that is used by the blood pressure monitors, and the phones they connect to, to communicate with one another. The technology was designed to be cheap, have a low power consumption and would be used to transmit small packets [25].

The technology operates on the same frequency as traditional bluetooth but has a much lower power consumption by remaining in sleep mode when not connected to anything.

There are three main pairing mechanisms with BLE: Just Works, Passkey and Out of Band. Just Works is a mechanism where no authentication takes place between the devices which makes it quite vulnerable to man-in-the-middle attacks. Passkey is a method where one device will display a key that must be entered into the other device for a pair to be successful. This method makes it more secure to mitm attacks. The final method is Out of Band where another technology is used to pass the key between devices [38].

The pairing process between BLE devices can be broken down into three phases. First, one device will send a pairing request to the other. They will trade their authentication process, IO capabilities and other information about the device. All of this information is unencrypted.

The second phase consists of the devices exchanging or generating keys for authentication. Then encryption keys are generated and implemented.

The optional third phase is where transport specific keys are shared between devices.

5.1.2 HTTP

The Hypertext Transfer Protocol is the protocol used to transmit data between the phone and the server over the internet. Most communication on the internet now uses HTTPS, an extension of HTTP which uses Transport Layer Security to encrypt all data that uses the protocol. Without HTTPS, anyone on the same network would be able to read the data being transmitted in clear text.

An app or website not using HTTPS would be a great security concern to all of its users.

5.2 Bluetooth pairing

Using the nRF connect app on Android, I was able to attempt to connect and bond with the blood pressure monitors without any authentication. Being able to connect to the monitors without authentication would allow an attacker to prevent the user from connecting the monitor to their phone and open the window for the attacker to gain access to blood pressure readings. This would create a denial of service, as the user would not be able to upload their readings.

The MocaCuff and app use a *Just Works* method to pair devices. The monitor cannot bond with any device, so the MocaCuff does not store the device they were previously connected to and the same pairing process must be followed any time the user wants to transfer readings from the monitor to the phone. To prepare the phone to pair with a MocaCuff device, a user would open the app and click the MocaCuff icon. The MocaCuff can be placed into pairing mode either by pressing the pairing button on the device or it will automatically enter pairing mode after it finishes taking a reading.

Once the MocaCuff and phone detect each other, the device immediately starts sending the phone the readings it has stored, without any authentication process.

The phone can be left in pairing mode indefinitely, opening the window for an attacker to leave a phone in pairing mode nearby someone they know to own a MocaCuff. This would allow the attacker to download all of the user's previous blood pressure readings and, while remaining connected to the monitor, a user would be unable to connect their phone to the device, creating a denial-of-service.

The Evolv monitor also uses a *Just Works* pairing method. The device has to be initially paired with the phone and, after the initial pairing, the devices bond with each other and will remember what device they were previously paired with. Though, the monitor can be connected to without needing to be authenticated or being bonded - doing so will prevent the user's smartphone from being able to connect to the monitor.

The Activ8 blood pressure monitor requires it to be connected to the phone while it is taking the reading, but it too uses *Just Works* and bonds with the phone it connects to. Requiring the device be connected to the phone while a reading is being taken ensures the user is actively aware that the monitor is connected to the correct device. I was not able to connect to the Activ8rlives blood pressure monitor with nRF connect. Despite my efforts, the monitor would only connect to the bonded phone using the Activ8 app. This prevents an attacker launching a denial-of-service attack on the user.

As all three devices use a Just Works pairing method, none of them are very secure. But, the MocaCuff is by far the least secure due to its lack of bonding with the phone and the fact that the phone can be left in pairing mode indefinitely.

5.3 Access Token

When sending requests to the web API, all three devices include an access token in their message which is sent to the phone at login - this can be seen in Figure 5.1. The Moca app and Activ8 access tokens have no expiry and changes only when the user changes their password. If an attacker were to gain knowledge of this access token then it could be used to send requests to the server indefinitely, until the user changes their password.

The Omron app uses the kii REST API [16] which implements the OAuth2 authorisation protocol [23]. The app receives a refresh token along with it's access token at login. The refresh token is used to get a new token once the original one expires after 60 minutes, this process is shown in Figure 5.2. This is considerably more secure than Moca and Activ8's apps as, even if an attacker gained knowledge of the access token, it would only be functional for the remainder of its life. Although, if an attacker were already familiar with web API for the Omron app, then they could still steal sensitive user information within that 60 minute window.

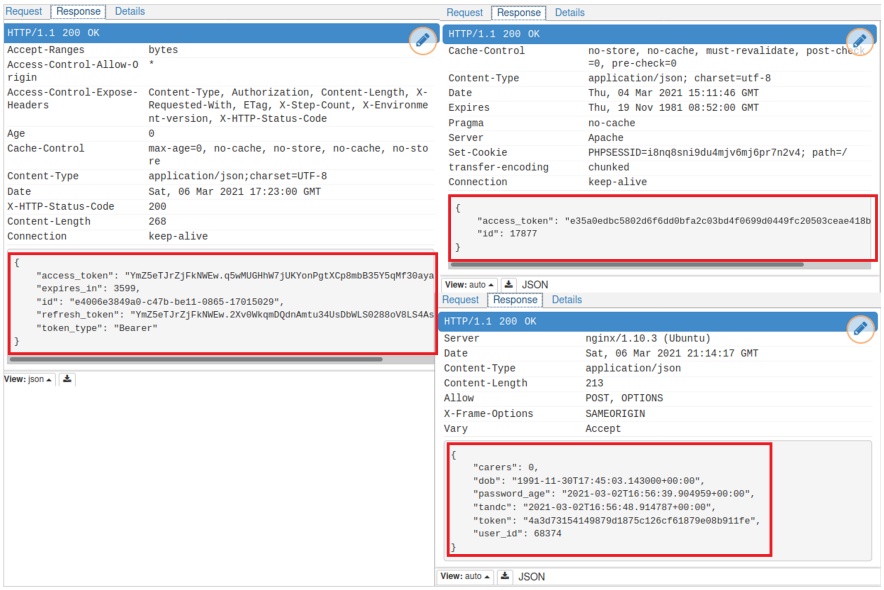

Figure 5.1: Receiving access tokens: **Left** - Omron; **Top Right** - Moca; **Bottom Right** - Activ8

5.4 Account creation and login

For all 3 devices, I used my Protonmail account to register for their services. They all asked for a name, email and password to create an account.

Figure 5.3 displays the process for all 3 devices. They transmit the login details as plain text over HTTPS for both account creation and login. The servers then respond with the access tokens. A man-in-the-middle attacker would be able to view all of this information via SSL stripping. With access to both an email and password, the attacker may be able to use this information to gain access to some of the user's other accounts as they may use similar passwords.

5.5 Profile

When creating an account a number of personal information is requested from all 3 apps, such as name, address, phone number and medical conditions. Once this information is turned over it is then sent to the server along with the access token. When logging in, this information is pulled from the server using the access token. The apps all regularly download the profile information to ensure they are up to date.

An attacker could intercept this information via a man-in-the-middle attack. This personal information could be used for identity theft, as all that is required to steal another's identity is their name, address and date of birth [27], leading to financial loss for the user.

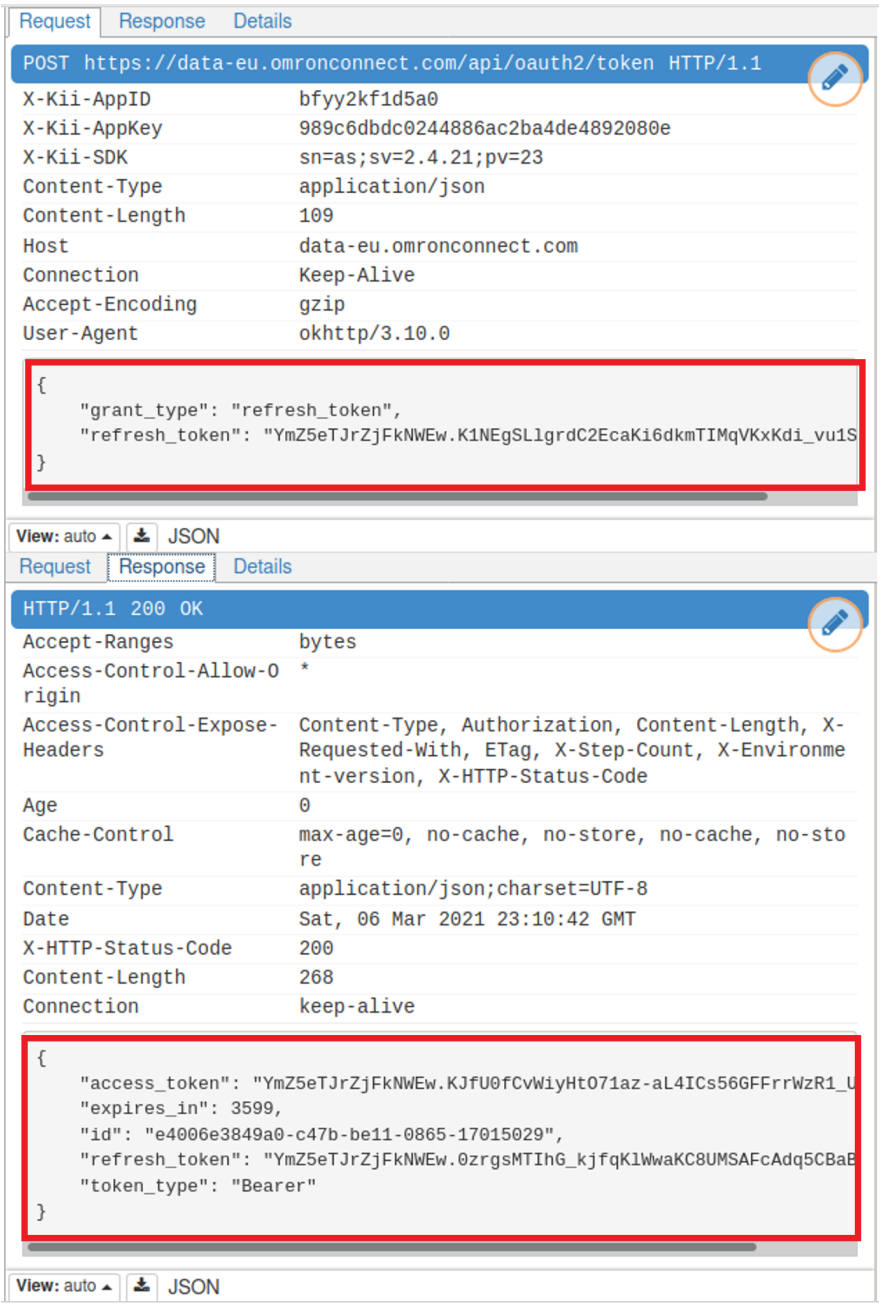

Figure 5.2: Omron connect app refreshing the token

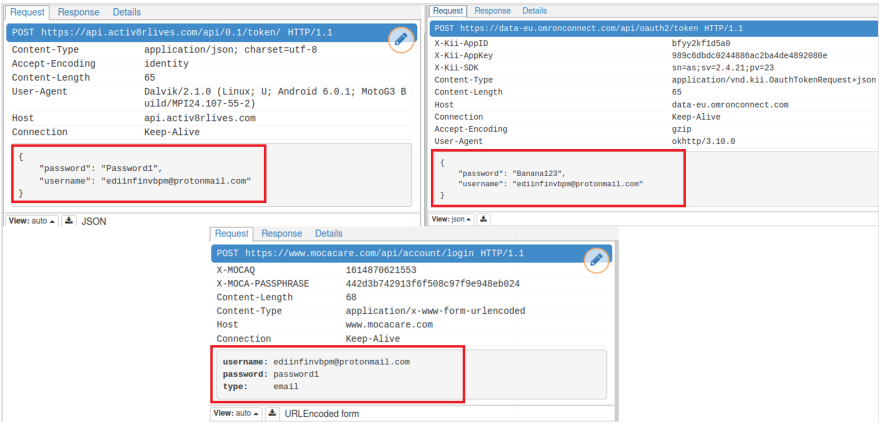

Figure 5.3: Login POST request: **Top Left** - Activ8; **Top Right** - Omron; **Bottom** - Moca

5.6 Blood Pressure Readings

After login, the devices request any blood pressure readings that were stored on the server; these readings are then stored on the phone. As before, this information could easily be read by an attacker using the access token or via a man-in-the-middle attack.

Upon closer inspection of the data sent between the Moca app and server, I found that the app was collecting GPS location data and uploading it along with the blood pressure readings, this can be seen in Figure 5.4. This information is not available within the app, nor is it disclosed within the privacy policy. The app requests access to location data as it says it is required to be able to connect with Bluetooth devices, this is shown in Figure 5.5. While this is true, the uploading of location data to Moca servers is not necessary for Bluetooth to function correctly. This raises a number of issues, namely that this undisclosed collection of personal information may be illegal and that this could be stolen by an attacker with access to the user's account without the user even knowing it is being collected. If an attacker gained knowledge of a user's access token for the Moca app then they could continually monitor the user's location. In Chapter 7, I discuss how I was able to compromise this information for the thousands of accounts who use the MocaCare service.

While testing how the MocaCare REST API requests work, I attempted to omit the access token in various requests to see if any would go through to the server successfully. While most requests returned an error, as the access token was incorrect, I did find that **a request for blood pressure readings with the access token omitted would succeed.** When the phone requests blood pressure readings from the server it sends a hash of the user's account ID and the access token, but a request with only the hash of the account ID will also succeed. If an attacker became aware of this hash then they would be able to access a user's blood pressure readings indefinitely, even if the user changed their password. This has very serious security concerns as a user would not even be aware that this information was being accessed. Upon further investigation I found that account IDs were generated sequentially and that the hash was generated locally - this is detailed further in Chapter 7.

Request Response Details

HTTP/1.1 200 OK

Cache-Control	no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Content-Type	application/json; charset=utf-8
Date	Thu, 04 Mar 2021 15:11:52 GMT
Expires	Thu, 19 Nov 1981 08:52:00 GMT
Pragma	no-cache
Server	Apache
Set-Cookie	PHPSESSID=fmbb4t9m9bfnjqn7naot2fe2j4; path=/
X-Paginator-Current-Page	1
X-Paginator-First-Page	1
X-Paginator-Last-Page	1
transfer-encoding	chunked
Connection	keep-alive


```
[
    {
        "created_at": "2021-01-21 01:19:27",
        "dbp": 86,
        "hr": 86,
        "id": 1079030,
        "lat": 38.8977,
        "lng": 77.0365,
        "note": null,
        "sbp": -999.99,
        "status": 1,
        "time": 1611160000,
        "timezone": "GB"
    },
```


Figure 5.4: Location data stored with blood pressure reading in Moca app

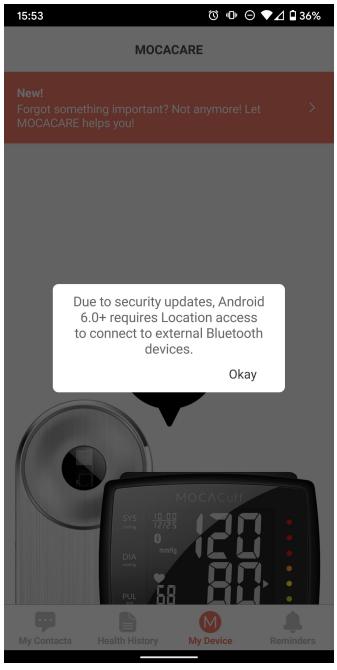

Figure 5.5: The Moca App requesting access to location data

5.7 Moca app additional features

The Moca app comes with a messaging and reminders feature.

Regarding the reminders feature, the application allows reminders to be set for doctors appointments, blood pressure readings and taking medication. A lot of this information is quite sensitive and can be accessed with access token. An attacker with access to this information could read what medication the user is taking, when they are visiting the doctor, what notes were taken at their doctors appointment and also delete, edit and create reminders that would be displayed to the user. The modification of data is discussed more in Chapter 6. Consequences of this could be dangerous, as the user may rely on the reminders feature to take vital medication - if the the dosage or frequency of these medication reminders were altered then this could cause harm to the user.

The messaging service allows users to send text, image and audio based messages to other users of the app, as well as share their blood pressure readings and account information. One security flaw with this service is that when two users add each other as contacts, they then each get access to each other's blood pressure readings, without either being informed of this fact. All of the user's contacts and messages could be intercepted via a man-in-the-middle attack or with knowledge of the access token.

The messaging service has a further, more serious flaw which allows an attacker to access the personal information of the thousands of accounts of the MocaCare service - this is discussed in detail in Chapter 7.

5.8 Website

When I started this project I first went to the MocaCare website to create an account, but was met with a page not found error when I attempted to do so. I returned later in the academic year to find the login service was now running but used HTTP and all login information was sent in clear text, as well as all account information, including blood pressure readings. If a user were to attempt to sign into their account then their email address and password would be clearly visible to anyone who was monitoring the network they were connected to - this would be of particular concern on a public WiFi network. An attacker with knowledge of the user's email address and password could use this information to try and access other accounts the user may have.

After checking the MocaCare website login throughout the year, I have found it to occasionally not be operational - consequently, the only way for a user to view their account information is via the app on a smartphone.

The Activ8 and Omron websites both use HTTPS to transmit data, but could still be vulnerable to man-in-the-middle attacks with SSL stripping to steal data.

Chapter 6

Injection of Data

These blood pressure monitors present an avenue for a user to monitor their health from day-to-day, in the comfort of their home. Many users rely on these devices to notify them early of any health concerns and they use the accompanying applications to remind them to take readings, medication and visit their doctor. Any manipulation of this data is not only a privacy violation, but could pose a serious health risk to the users of these devices and apps.

In this chapter I look at various methods an attacker can use to manipulate, create and delete user information that they may have stored on the monitors, phone or servers.

6.1 Manipulating app–server requests

mitmproxy not only allows for the interception of data, but also to alter it before it reaches the receiver. This means that an attacker could change data before it reaches the server or phone.

For the Activ8 and Moca apps, information could be easily changed within mitmproxy. I was able to alter blood pressure readings, account information and messages going to and from the server and phone. If the user's blood pressure readings were being monitored by another individual, such as a relative, then manipulating the blood pressure readings before they reach the server could raise a false alarm and create panic.

Omron's app and server was able to detect when data was altered and the server would return an error whenever I attempted to do so. This was due to a cyclic redundancy check that is built into the messages sent to the server.

6.2 Creating server requests

cURL is a command line tool that allows the user to send requests using various network protocols. In my case, I used it to craft GET and POST requests with the various servers the blood pressure monitor apps used.

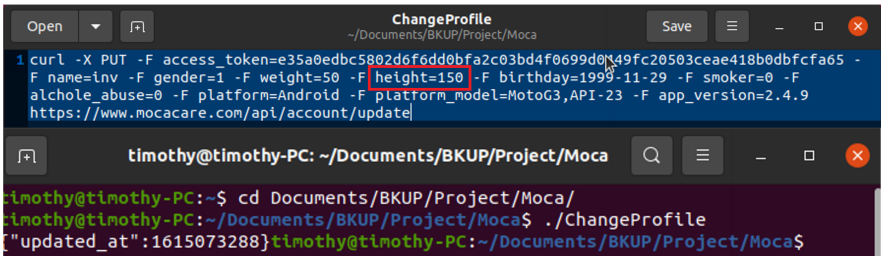

Figure 6.1: Crafting a message in cURL to change a user's height in the Moca App

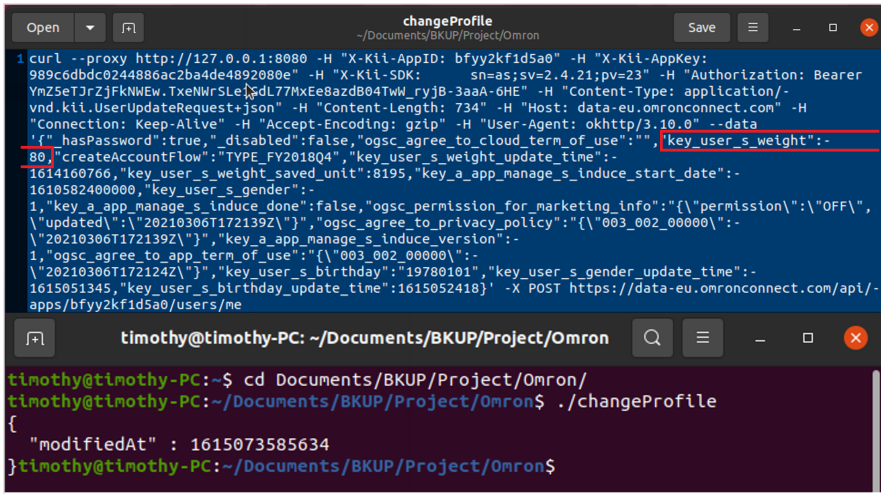

Figure 6.2: Crafting a message in cURL to change a user's weight in the Omron App

For all three apps I was able to craft messages that the server would accept. This includes Omron's app, which rejected my requests when I altered them in mitmproxy. I was able to craft fake blood pressure readings, alter account information at will and add various reminders and messages. These can be seen in Figures 6.1-6.3.

The ability to send fake readings and reminders to the server for these devices could ultimately lead to a denial of service as the device could be flooded with fake data, ultimately hiding all of the useful readings that the user has uploaded. This attack could also be used for more malicious purposes, as a user could be sent fake messages, medication reminders and blood pressure readings causing confusion, paranoia and possibly leading to the incorrect medication being taken.

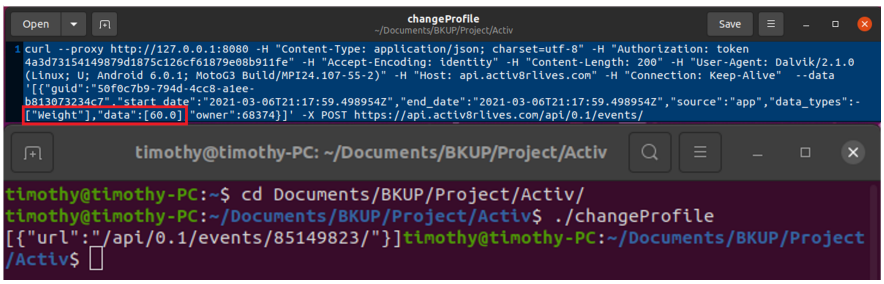

Figure 6.3: Crafting a message in cURL to change a user's weight in the Activ8 App

Chapter 7

A Closer Look at MocaCare

During my security analysis of the BPMs I discovered a number of flaws withing the MocaCare Web API which, upon investigating further, gave anyone the ability to access personal and medical information of the entire MocaCare user base. In this chapter I detail my discovery of these flaws and how I was able to use them to access the data of thousands of users.

7.1 Account ID

While analysing the communication between the Moca app and server, I created a number of accounts to test different features of the app. After I had created a few accounts I noticed that my account ID was not being randomly generated and was increasing by one with every account I made i.e. **the 7000th account would have the account ID 7000.** This allowed me to easily predict all of the account IDs on the service.

In the Moca app, when the device requests previous blood pressure readings from the server, the access token is sent along with a hash of the account ID. But, upon closer inspection of the information sent to the phone from the server, I could not find a single mention of the hash of the account ID. Therefore, I concluded that **the hashing must occur within the app, on device.** I will detail how I found the hashing algorithm in section 7.2.

While experimenting with changing the data that was sent to the server in the various GETs and POSTs that the phone sent, I found that the server does not check the access token when requesting blood pressure reading from the server. Therefore, **all that is required to gain access to the blood pressure readings is the hash of the account ID.** You will see in Figure 7.1 that both requests succeed, one with the access token and one without. Due to the account IDs being generated sequentially, having access to the hashing algorithm would allow an attacker to access the blood pressure readings of every user.

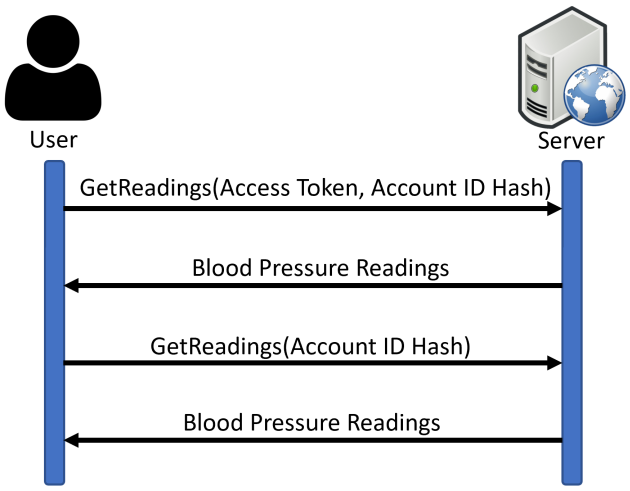

Figure 7.1: Both requests succeed: **First request** - Requesting readings with access token; **Second request** - Requesting readings without access token

7.2 Reverse engineering the Android app

I went in search of the account ID hashing algorithm within the MocaCare app's source code. To do this, I installed APK Extractor on my Android device and downloaded the APK file to my computer. I then used Visual Studio code with the APKLab plugin to decompile the APK to Java and find where the hashing algorithm was located. This took some time as I did not have any prior experience with Android development, nor any reverse engineering experience of source code. Also, the app source files contained hundreds of Java class files, a lot of which had been obfuscated making it difficult to understand what many of the methods were doing.

After locating the hashing algorithm, I initially altered the MocaCare app to download the blood pressure readings of the account I specified. I then recompiled it and installed it on the phone - when I launched the app, another account's blood pressure readings would be displayed instead of the readings belonging to the account that was logged in. After being sure I had understood the hashing algorithm, I created my own Java application with the hashing algorithm class along with any classes it depended on. I then fed in thousands of account IDs into the application and saved the resulting hashes to a comma delimited file.

7.3 Accessing blood pressure readings and location data

Due to the fact that the app secretly uploads location information with its blood pressure readings, gaining access to one would guarantee the other. As the app only requires the hash of the account ID to gain access to an account's readings, I sent my hashes to the server in individual GET requests and saved the responses.

Due to the large volume of accounts, manually sending thousands of requests to the

Figure 7.2: Information of a user that an invite has been sent to

server would be impossible to complete in a reasonable time frame. Therefore, I created a Python script, with the PycURL library, and created a loop over the hashes file that would send the GET requests to the server and save the data it replies with in individual files associated with each account, i.e. the file 7001.txt contained the blood pressure readings for the account number 7001.

7.4 Accessing email addresses, usernames and profile pictures

The messaging service of the MocaCare app is one of the last features I tested. I used two different Android phones, each with separate MocaCare accounts, to test the feature. I found that when a user invites another user to be a contact, personal information of the invited user is sent to the invitee's phone, but this is not displayed within the app, this information is shown in Figure 7.2. The process of adding a contact is briefly detailed in the next paragraph and in Figure 7.3.

When a user, say User A, attempts to add another user, say User B, as a contact User A will do so using the email address User B used to create their account. When User A sends this email to the server, it replies with User B's account number which is then sent in an invite POST request to the server. The server then sends User B an invite to accept User A as a contact. As soon as User A sends the invite to User B, User B's information is added to User A's contacts, regardless of whether User B accepts the invitation or not. User B's contact information includes their email address, username and profile picture, all of which is downloaded when the app requests User A's contact list.

To gain access to the email addresses, usernames and profile pictures of all accounts on the service I skipped the email look-up step and just sent an invite to all account IDs, using a similar Python script as I used to obtain blood pressure readings, which then added all of their information to my contacts list. I then saved the resulting con-

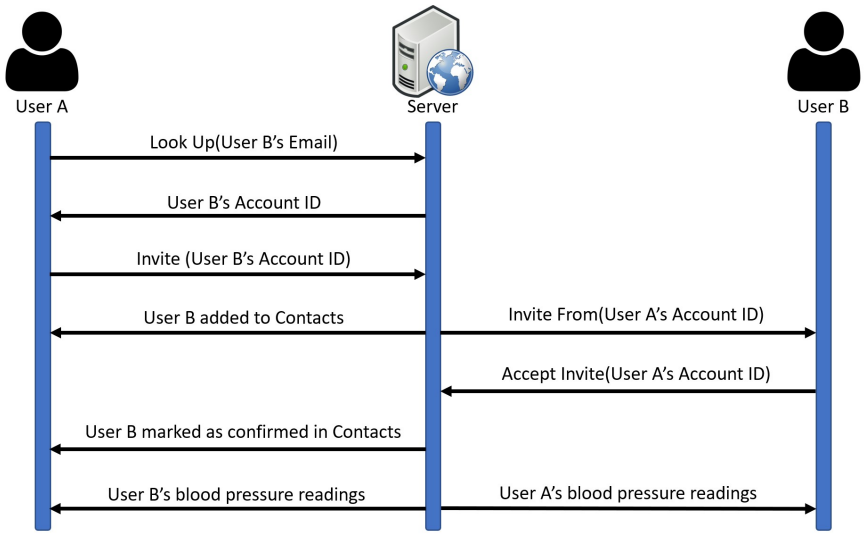

Figure 7.3: Exchange between devices and server when User A invites User B to be a contact

tacts list and matched up the account IDs from this information with the account IDs of the blood pressure readings. Ultimately, I had the email addresses, usernames, profile pictures, blood pressure readings and location information of all accounts on this service.

Chapter 8

Results analysis

Vulnerability	Moca	Evolv	Activ8
Just Works pairing	✓	✓	✓
Connection to unbound devices	✓	✓	-
Unauthorised bonding	-	✓	-
App pairing mode has no timeout	✓	-	-
Monitor denial of service	✓	✓	-
Email address sent in clear text	✓	✓	✓
Password sent in clear text	✓	✓	✓
User profile sent in clear text	✓	✓	✓
Token sent in clear text	✓	✓	✓
Token has no expiration	✓	-	✓
No certificate pinning	✓	✓	✓
Manipulate REST requests in transit	✓	-	✓
Manipulate user data	✓	✓	✓
All user data vulnerable	✓	-	-
On device hashing	✓	-	-
Send custom messages	✓	✓	✓
Website is in HTTP	✓	-	-
Undisclosed location tracking	✓	-	-
Access to account information of all users	✓	-	-

Figure 8.1: Summary of all identified vulnerabilities. ✓, Vulnerability found; -, Vulnerability not found

8.1 Comparison

All devices have been shown to have security vulnerabilities that need addressing, although some are more serious than others.

Starting with the BPMs pairing with the phone, MocaCuff showed the most severe vulnerability as the device did not bond with the phone and would transmit its data to the first phone it detected. Omron's Evolv is the more secure as it bonds with the phone and will automatically reconnect to it after a reading has taken place. But, unauthorised devices can connect to it, preventing it reconnecting to the bonded device. Activ8's device is the most secure because it stays connected to the phone as it takes a reading, allowing the user to see if the phone is connected to the monitor before they take a reading. Also, unauthorised devices cannot connect to it, preventing denial-of-service attacks.

In terms of access tokens, Moca and Activ8 are equally insecure as their tokens have no expiry and only change when the user changes their password. Omron's app token has an expiry of one hour and sends a refresh token to the server to obtain a new access token.

All three devices are equally vulnerable to man-in-the-middle attacks to steal the user's login credentials at account creation or login. A potentially more secure method would be to hash the credentials before transmission.

All information associated with a user's account is transmitted in plain text over HTTPS, and is also regularly refreshed by the app. This makes this information vulnerable to a mitm attack. Information such as medication, doctors appointments and messages with other people could be stolen.

One severe privacy and security concern is the collection of a user's location information by the Moca app. This collection of information is not disclosed to the user in the Privacy Policy and is transmitted in plain text - this is a clear violation of a user's privacy.

The Moca website had a concerning security flaw as its login was in HTTP, meaning a user's login information would be transmitted in clear text and be unencrypted, allowing anyone along the connection between device and server to view this information. The Activ8 and Omron websites both used HTTPS.

Activ8 and Moca's traffic between app and server could both be manipulated, altering the data going between server and phone. Omron's service was able to detect any changes to data being transmitted. Although, all three devices were susceptible to having entirely new messages being sent to their servers, all that was required was a user's access token and then crafting the message was relatively trivial.

The most severe vulnerability found was the ability to access most of the account information of all of the users of the MocaCare app. This was caused by account IDs not being randomly generated, the IDs being hashed on device and blood pressure readings only requiring the hash to be accessed. No wide scale breach of the other two devices could be found.

8.2 Implication

Here I will discuss the implications of these security vulnerabilities on the user and the services, the skills and equipment an attacker may need to launch these attacks and how likely it is for these attacks to take place.

The most severe security flaw found was the ability to access all accounts of the MocaCare service. This would **allow an attacker to access blood pressure readings, location data, usernames, nicknames, profile pictures and email addresses of all of the thousands of users of the MocaCare app.** The attacker could be anywhere in the world and would not have to be near anyone who actually owned a MocaCuff or used the MocaCare app. With access to this information, an attacker could potentially steal a user's identity, resulting in significant monetary loss for the user. The attacker would need to be thoroughly familiar with the the Web API that MocaCare uses and the vulnerability requires some trial-and-error to uncover, so it is very unlikely most people would find it. But, once uncovered, **the code to download the information is only about 20 lines long** and it would only require one individual for this information to be uploaded publicly to the internet.

Another severe security and privacy vulnerability found is the **collection of location information by the MocaCare app without this being disclosed to the user** in the privacy policy or within the app. This is a clear violation of a user's privacy and is likely illegal. This fact would not be easy to uncover for an average user, as the location information can only be seen using software and a set-up that allows an individual to monitor the requests going to and from the phone and server.

The lack of expiry of access tokens in the Activ8 and Moca apps is a high security risk as this token is transmitted with every POST and GET transmission to the server. If an attacker were to gain knowledge of a user's token then they **could continually monitor a user's activity and manipulate, create and delete data on a user's account without even having to know their password.** The user would have no way to know there was a breach in their account unless they noticed strange changes being made. The only way to boot an attacker would be to change their password. To actually gain knowledge of this access token the attacker would have to perform SSL stripping of a user's traffic which would require the user to have fake CA certificates installed on their device. The potential difficulty of achieving this makes the risk less severe. Once an attacker has succeeded at gaining knowledge of the access token making use of it is very easy though and the attacker would be able to use it anywhere in the world, regardless of where the victim was located.

The Moca website account page using HTTP is another high risk security concern. A user's login information as well as account information could be compromised as their traffic would be transmitted unencrypted. For an attacker to avail of this vulnerability they would have to be monitoring network traffic anywhere along the line between device and server. If the user were to be using public WiFi this would put them at particular risk as no specialist equipment would be required to see the user's information except some software such as Wireshark.

All apps are vulnerable to having fake REST requests being sent to the web API. This

would be considered a high risk because of the sensitive nature of the information. An attacker could craft fake medication reminders, alter medication doses or delete reminders all together. This is potentially life threatening. An attacker could also launch a denial-of-service attack by flooding a user's account with spam messages, making it almost impossible for a user to use the app. An attacker would have to gain knowledge of the access token to perform this attack, which makes this quite difficult to perform.

A medium risk vulnerability is the pairing mechanism MocaCare uses. Any phone with the Moca app installed could be left in pairing mode indefinitely and would pair with a MocaCuff as soon as it starts searching for a phone. The phone would then have access to all blood pressure readings that device has previously taken. An attacker would have to know of someone who owns a MocaCuff and would also have to place a device near to the user when they are taking a reading. The difficulty of this task is why this is considered a medium vulnerability. If an attacker were aware that a user owned one of these devices, then performing the attack would be almost trivial - the attacker would only need to download the MocaCare app, leave it in pairing mode and wait.

A low risk vulnerability is Omron Evolv's pairing mechanism. An attacker who was in close proximity to a user would be able to connect to the monitor and prevent the user connecting their phone to it. This prevents the user from downloading their blood pressure readings to their phone but, the attacker would not be able to read the blood pressure readings from the monitor. This is considered to be of low risk due to the attacker needing to be near the victim, the data not being able to be stolen and the fact that the attacker would need to be using a Bluetooth debugging app or device to launch it.

Chapter 9

Recommendations

This chapter will detail the recommended steps that should be taken to mitigate the security vulnerabilities detailed in this report.

MocaCare API flaw
To address the MocaCare data breach issue a few steps should be taken. Firstly, the access token should be verified when attempting to send a GET request for blood pressure readings from the server. This will verify the user has actually received the access token but this does not combat attackers who have stolen the access token - a remedy to this is detailed below. Next, the account IDs should either be randomly generated or hashed so they can not be easily predicted. This will prevent invites being able to be sent to the entire user base and would also prevent an attacker using the built in hashing function to hash all user IDs. Thirdly, a user's information should not be added to the invitee's contacts unless the user has accepted the invite.

Undisclosed location information collection
The collection of location information within the MocaCare app should be ceased and all previously collected information should be deleted immediately. The location information was collected without the users' consent and MocaCare therefore has no right to store it. If MocaCare wishes to collect user location information then this should be clearly disclosed to the user and their consent obtained before any data is stored.

Access token generation
A system similar to Omron's OAuth2 authentication process should be implemented on Moca and Activ8's app. This ensures that, even if the access token becomes compromised, that it will eventually expire and become useless. It is also recommended that the Omron app refreshes tokens more regularly than once an hour.

Certificate pinning
All companies should implement Certificate pinning within their apps to help prevent man-in-the-middle attacks. This would prevent attacks from using services such at mitmproxy to monitor traffic between phone and server. For an attacker to monitor traffic they would have to somehow install a modified version of the application on the victim's phone, making an attack a lot more difficult to carry out.

End-to-end encryption
End-to-end encryption could also be implemented between the phone and server to prevent man-in-the-middle attacks. This will ensure the user's data is secure when connected to untrusted networks.

Bluetooth pairing timeout
The MocaCare app should eventually time out when it is left in pairing mode for a considerable length of time - this will prevent an attacker from being able to leave a phone nearby and await the MocaCuff entering pairing mode.

Unauthorised Bluetooth pairing
To prevent unauthorised Bluetooth pairing, the MocaCuff should bond with the user's phone and then only connect with that phone in the future. This will allow smoother and more secure sharing of data between monitor and phone after the initial pairing has been complete. Also, the Omron Evolv should not let devices that have not been bonded to it connect to the monitor. This will prevent user's blood pressure readings being read by attackers and also prevent denial-of-service attacks.

HTTPS web login
The MocaCare website should use HTTPS throughout its site to ensure user login information, as well as account details, are kept secure. The site should also not go down for extended periods of time as users should have easy access to their account information.

Chapter 10

Conclusion

With the increase in popularity of wearable and IoT devices every year and, our society's further reliance on at home medical monitoring in the wake of the pandemic, security should be one of the highest concerns when developing these devices.

In this project I found a range of security vulnerabilities in 3 blood pressure monitors: MocaCare MocaCuff, Activ8rlives blood pressure monitor, Omron Evolv. The vulnerabilities I found range from mild to severe with the worst exposing personal information of thousands of people. Of the 3 devices, the Omron Evolv was the most secure, and with the device having the most users by far, with over a million, this is reassuring. The MocaCare Cuff was the least secure, with the service's thousands of users at risk of having their personal data being leaked. I launched man-in-the-middle attacks, denial-of-service attacks and manipulated and fabricated server requests.

Throughout my project I found 35 vulnerabilities across the 3 devices. I found a Web API flaw which would allow someone to access all of the personal and medical information of the entire MocaCare user base. I uncovered the fact that MocaCare have been uploading user location information to their servers without the user's consent. I also found that access tokens do not have an expiry in either of the MocaCare or Activ8rlives apps, allowing an attacker with knowledge of the token to have indefinite access to the user's account.

For these blood pressure monitors, there is still the potential for there to be vulnerabilities within the app source code that could be discovered in future research. Also, the blood pressure monitor hardware itself could be examined to attempt to find avenues for attack.

There are many blood pressure monitors, like the devices I was investigating, on the market, leaving the potential for many security flaws to be present. With the little research into devices like these available prior to my project, it is clear that this is a market that should be studied significantly more.

Bibliography

[1] Common methodology for information technology security evaluation. https://www.commoncriteriaportal.org/files/ccfiles/cemv10.pdf, Aug 1999.

[2] Nist releases draft guidance on internet of things device cybersecurity. https://www.nist.gov/news-events/news/2020/12/nist-releases-draft-guidance-internet-things-device-cybersecurity, 2020.

[3] Application threat modelling. https://owasp.org/www-community/Application_Threat_Modeling, 2021.

[4] Proton Technologies AG. Protonmail. https://protonmail.com/, 2021.

[5] Fadi Al-Turjman, Muhammad Hassan Nawaz, and Umit Deniz Ulusar. Intelligence in the internet of medical things era: a systematic review of current and future trends. *Computer Communications*, 150:644–660, 2020.

[6] Zainab Alansari, Nor Badrul Anuar, Amirrudin Kamsin, Safeeullah Soomro, and Mohammad Riyaz Belgaum. The internet of things adoption in healthcare applications. In *2017 IEEE 3rd International Conference on Engineering Technologies and Social Sciences (ICETSS)*, pages 1–5. IEEE, 2017.

[7] Steve Alder. Ponemon: 89 percent of healthcare organizations have experienced a data breach. https://www.hipaajournal.com/ponemon-89-pc-healthcare-organizations-experienced-data-breach-3430/, May 2016.

[8] Steve Alder. 87% of healthcare organizations will adopt internet of things technology by 2019. https://www.hipaajournal.com/87pc-healthcare-organizations-adopt-internet-of-things-technology-2019-8712/, Mar 2017.

[9] Thelma Allen. Nist cybersecurity for iot program. https://www.nist.gov/programs-projects/nist-cybersecurity-iot-program, Oct 2020.

[10] Faisal Alsubaei, Abdullah Abuhussein, and Sajjan Shiva. Security and privacy in the internet of medical things: taxonomy and risk assessment. In *2017 IEEE 42nd Conference on Local Computer Networks Workshops (LCN Workshops)*, pages 112–120. IEEE, 2017.

[11] Taylor Armerding. Medical devices at risk: 5 capabilities that invite danger. https://www.csoonline.com/article/3202081/medical-devices-at-risk-5-capabilities-that-invite-danger.html, Jul 2017.

[12] Nordic Semiconductor ASA. nrf connect. https://play.google.com/store/apps/details?id=no.nordicsemi.android.mcp&hl=en, 2020.

[13] Ananya Bajaj, Meghna Bhatnagar, and Anamika Chauhan. Recent trends in internet of medical things: a review. *Advances in Machine Learning and Computational Intelligence*, pages 645–656, 2021.

[14] Jet Brains. Intellij idea. https://www.jetbrains.com/idea/, 2021.

[15] Jiska Classen, Daniel Wegemer, Paul Patras, Tom Spink, and Matthias Hollick. Anatomy of a vulnerable fitness tracking system: Dissecting the fitbit cloud, app, and firmware. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2(1):1–24, 2018.

[16] Kii Cloud. Kii rest api guide. https://docs.kii.com/en/guides/thingifsdk/non_trait/rest/.

[17] Culture Department for Digital. Etsi industry standard based on the code of practice. https://www.gov.uk/government/publications/etsi-industry-standard-based-on-the-code-of-practice, journal=GOV.UK, Jul 2020.

[18] ENISA. Good practices for security of iot - secure software development lifecycle. https://www.enisa.europa.eu/publications/good-practices-for-security-of-iot-1, May 2020.

[19] Hossein Fereidooni, Jiska Classen, Tom Spink, Paul Patras, Markus Miettinen, Ahmad-Reza Sadeghi, Matthias Hollick, and Mauro Conti. Breaking fitness records without moving: Reverse engineering and spoofing fitbit. In *International Symposium on Research in Attacks, Intrusions, and Defenses*, pages 48–69. Springer, 2017.

[20] Hossein Fereidooni, Tommaso Frassetto, Markus Miettinen, Ahmad-Reza Sadeghi, and Mauro Conti. Fitness trackers: fit for health but unfit for security and privacy. In *2017 IEEE/ACM International Conference on Connected Health: Applications, Systems and Engineering Technologies (CHASE)*, pages 19–24. IEEE, 2017.

[21] Python Software Foundation. Python 3.9. https://www.python.org/, 2021.

[22] Google. Android debug bridge. https://developer.android.com/studio/command-line/adb, 2021.

[23] IETF OAuth Working Group. Oauth 2.0. https://oauth.net/2/, 2012.

[24] George Hatzivasilis, Othonas Soultatos, Sotiris Ioannidis, Christos Verikoukis, Giorgos Demetriou, and Christos Tsatsoulis. Review of security and privacy for the internet of medical things (iomt). In *2019 15th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, pages 457–464. IEEE, 2019.

[25] Robin Heydon and Nick Hunn. Bluetooth low energy. *CSR Presentation, Bluetooth SIG*, 2012.

[26] Chin-Lung Hsu and Judy Chuan-Chuan Lin. An empirical examination of consumer adoption of internet of things services: Network externalities and concern for information privacy perspectives. *Computers in Human Behavior*, 62:516–527, 2016.

[27] Ico.org.uk. Identity theft. https://ico.org.uk/your-data-matters/identity-theft/, 2021.

[28] Dave Johnson. The 7 best blood pressure monitors that you can use at home. https://www.forbes.com/sites/forbes-personal-shopper/2021/01/01/best-blood-pressure-monitor-2021/?sh=242230a53991, 2021.

[29] Francesco Lamonaca, Eulalia Balestrieri, Ioan Tudosa, Francesco Picariello, Domenico Luca Carnì, Carmelo Scuro, Francesco Bonavolontà, Vitaliano Spagnuolo, Gioconda Grimaldi, and Antonio Colaprico. An overview on internet of medical things in blood pressure monitoring. In *2019 IEEE International Symposium on Medical Measurements and Applications (MeMeA)*, pages 1–6. IEEE, 2019.

[30] Haoyu Liu, Tom Spink, and Paul Patras. Uncovering security vulnerabilities in the belkin wemo home automation ecosystem. In *2019 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pages 894–899. IEEE, 2019.

[31] Aseptika Ltd. Activ8rlives health+wellness. https://play.google.com/store/apps/details?id=com.activ8rlives.activ8rlives4&hl=en_US&gl=US, 2020.

[32] Canonical Ltd. Ubuntu 20.04. https://ubuntu.com/download/desktop, 2020.

[33] Microsoft. Visual studio code. https://code.visualstudio.com/, 2020.

[34] Rabbi Jason Miller. Best tech products of ces 2017. https://www.huffpost.com/entry/best-tech-products-of-ces-2017_b_58bafc8ee4b02eac8876cf3b, Mar 2017.

[35] MOCAheart. Mocacare. https://play.google.com/store/apps/details?id=com.atommedical.mocacare&hl=en&gl=US, 2020.

[36] Ltd. OMRON Healthcare Co. Omron connect. https://play.google.com/store/apps/details?id=jp.co.omron.healthcare.omron_connect&hl=en_GB&gl=US, 2021.

[37] OWASP. Owasp internet of things project. https://wiki.owasp.org/index.php/OWASP_Internet_of_Things_Project, 2018.

[38] Sode Pallavi and V Anantha Narayanan. An overview of practical attacks on ble based iot devices and their security. In *2019 5th International Conference*

on Advanced Computing & Communication Systems (ICACCS), pages 694–698. IEEE, 2019.

[39] Mitmproxy Project. mitmproxy. https://mitmproxy.org/, 2020.

[40] Oleg Pudeyev. Pycurl 7.43.0.6. https://pypi.org/project/pycurl/, 2020.

[41] Kristi Rawlinson. Hp study reveals 70 percent of internet of things devices vulnerable to attack. http://www8.hp.com/us/en/hp-news/press-release.html?id=1744676, Jul 2014.

[42] RIPE. iot-wg mailing list. https://lists.ripe.net/mailman/listinfo/iot-wg, 2020.

[43] sandrob. Drony. https://play.google.com/store/apps/details?id=org.sandrob.drony&hl=en_GB&gl=US, 2020.

[44] Shachar Siboni, Asaf Shabtai, and Yuval Elovici. Leaking data from enterprise networks using a compromised smartwatch device. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, pages 741–750, 2018.

[45] Internet Society. Iot security policy platform. https://www.internetsociety.org/iot/iot-security-policy-platform/, 2019.

[46] Surendrajat. Apklab. https://marketplace.visualstudio.com/items?itemName=Surendrajat.apklab, 2021.

[47] Inc. Synopsys. Synopsys and ponemon study highlights critical security deficiencies in medical devices. https://www.prnewswire.com/news-releases/synopsys-and-ponemon-study-highlights-critical-security-deficiencies-in-medical-devices-300463669.html, Jun 2018.

[48] Meher Tools. Apk extractor. https://play.google.com/store/apps/details?id=com.ext.ui&hl=en, 2020.

[49] Jirar Topouchian, Zoya Hakobyan, Jennifer Asmar, Svetlana Gurgenian, Parounak Zelveian, and Roland Asmar. clinical accuracy of the omron m3 comfort® and the omron evolv® for self-blood pressure measurements in pregnancy and pre-eclampsia–validation according to the universal standard protocol. *Vascular health and risk management*, 14:189, 2018.

[50] Nick Triggle. Nhs 'dangerously' short of 100,000 staff. https://www.bbc.co.uk/news/health-43143325, Feb 2018.

[51] Serge Truth. Create a threat model - step 1. https://blog.securityinnovation.com/blog/2011/02/create-a-threat-model-step-1.html, 2011.

[52] Vikas Vippalapalli and Snigdha Ananthula. Internet of things (iot) based smart health care system. In *2016 International Conference on Signal Processing, Communication, Power and Embedded System (SCOPES)*, pages 1229–1233. IEEE, 2016.