An Investigation into the Security and Privacy of Blood Pressure Monitors

Wei Tat Lee

Master of Science
Computer Science
School of Informatics
University of Edinburgh
2019

Abstract

In this paper, we show security vulnerabilities found commonly in connected healthcare devices. The lack of security practices in the production of these devices is a major concern for these devices before they become widespread in the healthcare sector. This paper illustrates that these security issues are well-known and should be prevented by the manufacturers, which is not the case in the majority of instances examined. This is important because Internet of Things (IoT) devices are increasingly used to monitor healthcare, and we see that security issues could become a major roadblock for these devices to be relied on. In this project, we build a testbed to apply penetration testing on the Activ8rLives Blood Pressure Monitor. We focus on analysing the security vulnerabilities that exist on the network communication of the system. We show that sensitive information is retrieved using MITM proxy. Besides, we also show that the application uses a non-expired token to authenticate users, which allows client replay attacks indefinitely. Denial of service attacks can also be made by using any device with Bluetooth Low Energy (BLE) capabilities to hog the connection with the monitor. This is due to the lack of authentication for the BLE communication. Next, we show that the smartphone application is not obfuscated. Lastly, we programmed a BLE-embedded device to spoof false measurements to the smartphone. We show that BLE embedded device can spoof 19 out of 20 measurements made by the user. We conclude that the blood pressure monitor itself provides the ease of connectivity, but does not employ the common security practices. We also provide a few suggestions to mitigate the exploits that we found in this paper. A comparative security study could be applied to different blood pressure monitor from manufacturers as future work.

Acknowledgements

I would first like to thank my project supervisor Dr. Paul Patras of the School of Informatics. I would also like to express my very profound gratitude to my friends for proof reading my thesis. And lastly I would like thank my family for the unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them.

Thank you.

Table of Contents

Chapter 1

Introduction

Statistics have shown the potential of *IoT* devices in the health care ecosystem through its increasing use in daily self-cared monitoring devices (DigitalHealth, 2018). These devices can be monitored remotely by health practitioners in real-time and are becoming more common in replacing the need for body checkups in public health facilities (IoT_Agenda, 2018; ReadWrite, 2018). For example, as the proportion of the world's population and risk of getting high blood pressure increases over time and age (WPP, 2017), digital blood pressure monitoring devices can assist health practitioners provide 'early' preventive actions for patients with the risk of heart attacks. Besides, digital blood pressure monitoring devices are now commonly used by elderly people.

Under those circumstances, the trust in these connected medical devices is needed inevitably (TheGuardian, 2019). However, having a system to be impenetrable has always been a big challenge in this modern society.

Due to the nature of strict deadlines, manufacturers often focus on releasing new features to compete in the market (Matt Toomey, 2018). This results in devices with known security flaws being released to the consumers (Tuptuk & Hailes, 2018). Nevertheless, user often rely on vendors for the protection of their data (Accenture, 2015), which in most cases are not enforced. Inevitably, the security aspects of the production are overlooked or neglected by the manufacturers.

With that in mind, there are an increasing number of cyber-attacks on these devices (Service, 2019) as open-source hardware platforms and drivers are becoming more prevalent (Scott Amyx, 2018) allowing individuals with minimal networking and programming skills to hack such commodity web-connected devices. Cyber attackers would try to exploit different kinds of vulnerabilities from unsecured devices to achieve malicious goals. For instances, an attacker could gain monetary benefits by

removing service availability from competitors.

One example of these attacks is the Mirai Botnet Attack (Symantec, 2016). It utilised unsecured IoT devices that were mainly configured to default factory settings to employ a Distributed Denial of Service (DDoS). Also, the user themselves could be applying the exploits themselves to gain monetary benefits. For example, NHS Scotland provides health services for hypertension patients (Scotland, 2019); this further motivates users with malicious intents to spoof fake blood pressure readings to gain these services.

Consequently, this led us to ask if we should trust these devices and constitutes a major roadbloack towards deploying *IoT* devices in critical public services, as long as their security remains under jeopardy. Therefore, we are going to identify the most common security vulnerabilities present in the particular case of blood pressure monitoring devices within this project.

The rest of this paper is organised as follows. **Chapter 2** presents related works and also provides detail information on background technologies. We then present the methodologies for our security analysis in **Chapter 3** which includes building a testbed and using various tools to assists in our analysis. Next, we present our results in **Chapter 4** on any security vulnerabilities on the Activ8lives Blood Pressure monitor. We then suggest some design recommendations to mitigate the attack that we found in **Chapter 5**. Finally, we summarise and make a conclusion of the project in **Chapter 6**.

Contributions and Findings

In this paper, we showed that the Activ8lives Blood pressure monitor is insecure. We can exploit the device using multiple attack vectors which are shown as follows:

- A testbed is built to perform comprehensive security testing on the devices.
- *MITM* proxy is deployed to sniff unencrypted data transmitted to the cloud server.
- False data is injected into the cloud server using replay attacks.
- The BLE communication protocol is reverse engineered.
- The application is decompiled and the source code is retrieved.
- A *DDoS* attack is applied to the smartphone application.

- An embedded device is programmed to spoof fake measurements which can be automated.

Responsible disclosure:

Prior to submitting this thesis, we have disclosed all of the identified security vulnerabilities to Activ8rLives.

Chapter 2

Background

2.1 Related Work

With *IoT* devices forecast as being a focus on enterprise attacks (iscoop.eu, 2018), heavy security analyses have been conducted on them. In particular, there is an extensive research targeting *Fitbit* devices, which is leading the market for health-tracking devices (Maher et al., 2017). Cyr et al. (2014) uncovered security weakness on the BLE and network traffic of FitBit Flex which includes static MAC addressing and exposed credentials during pairing. Furthermore, work has done by Fereidooni et al. (2017); Classen et al. (2018), which looked into the protocol for data transmission across the device, application and the cloud. In recent work, Orlosky et al. (2019) focused on the security and privacy concerns used by third-party services.

To the best of our knowledge, no security analyses that specifically targets blood pressure monitors devices has even been done. However, there have been more general studies on the vulnerabilities of medical devices, including devices that are not connected to the internet. In general, the recent work done by Yaqoob et al. (2019), conducted a study on hundreds of state-of-the-art networked medical devices and showed multiple vulnerabilities that exist in these devices.

All this research demonstrated how *IoT* devices, in general, contain security vulnerabilities that are common and attacks that are unique and highly specific. We look into these common exploits and apply them to blood pressure monitoring devices. In this paper, we focus on the communication protocol between the devices, the application and cloud services.

2.2 Bluetooth Low Energy

Bluetooth Low Energy (*BLE*) or short for Bluetooth Smart, is a short-range radio wave communication standard developed by Bluetooth SIG (SIG., 2019). BLE enabled devices simplify data transmission between two devices. In particular, its usage has shown to increase in the industry setting for *IoT* devices (Jeon et al., 2018). This is because *BLE* is designed to provide low power consumption. Most importantly, this greatly benefits *IoT* embedded devices, which normally are not powered with an AC/DC supply.

BLE uses the notation of profiles and services to define the functionality of the devices. The generic access profile (GAP), defines the base profile for devices to discover and connect. It also defines how the devices are paired with each other to have secured end-to-end encryption.

The data transfer between the devices after a connection is established is defined by the Generic Attribute Profile (GATT). Services are objects that contain multiple characteristics. These are used to organise larger data into a smaller chunk of data. The characteristic itself is the lowest level that contains a single data entity. Bluetooth *SIG* had officially defined a list of profiles that encapsulates services and characteristics that cover usual use cases. **Figure: 2.1** shows the predefined service for heart rate monitor devices.

GATT is built on top of the Attribute Protocol (ATT), which establishes how the data is being transported and stored. It formats data into services and characteristics which are encapsulated in a single GATT entity. Each of these objects is uniquely identified with a unique handle and a pre-defined 16-bit or 128-bit Universally Unique Identifiers (UUID).

2.3 Hypertext Transfer Protocol / Secure

HyperText Transfer Protocol Secure (HTTPS) is a secured communication protocol (Google, 2019) extended from HTTP (Fielding et al., 1999). It is built on top of the Transport Layer Security Protocol (TLS), which is a cryptographic layer that provides a secure communication across HTTP connections. In other words, a server and client that uses HTTP to communicate are encrypted end-to-end. This gives the assurance that an attacker with malicious intent would not be able to eavesdrop on sensitive information from the communication.

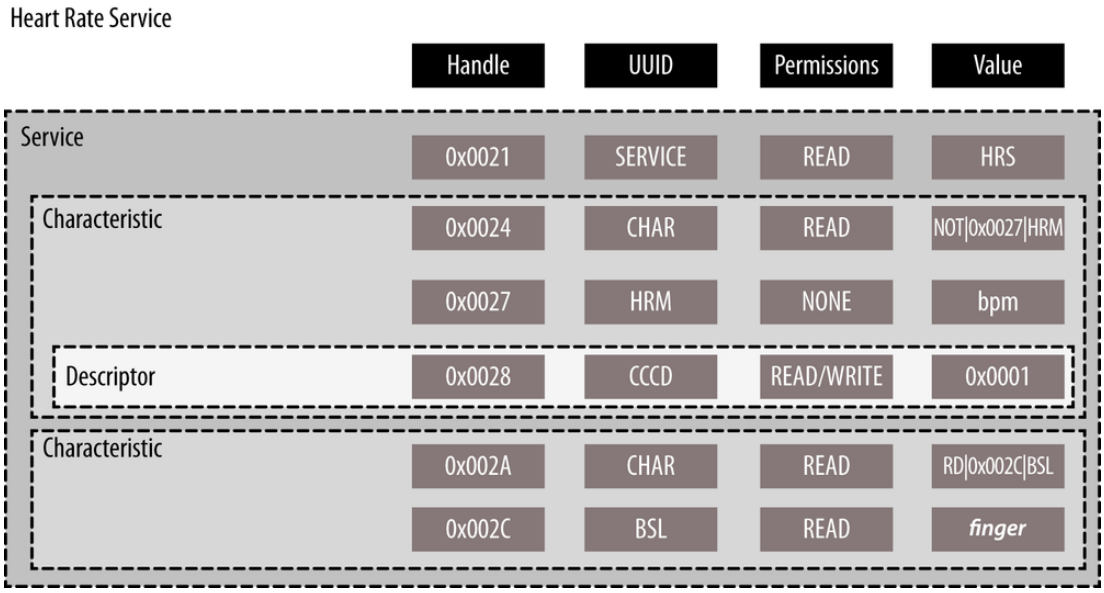


Figure 2.1: The Heart Rate Service Profile defined by Bluetooth *SIG*. (Davidson, 2014)

However, this can be circumvented when an attacker is in between the server and the client. The attacker tricks the client that they are the server and the server to be the client. This is known as the Man in the Middle (MITM) attack. This brings out the notion of digital certificates, which certify that the server itself is the rightful owner of the server. However, this can only be signed by a trusted Certificate Authorities (CA). Typically, client devices have a list of CA to be trusted, called root certificates. Thus, a malicious party would not be able to eavesdrop on the communication unless a self-signed certificate have been installed in the client's device.

2.4 Information Assurance

In this section, we will analyse the security aspects of the system based on information assurance.

Information assurance is defined as a set of processes or practices that ensure that information is processed or managed properly. The processes include the assurance of confidentiality, integrity, availability, authentication and non-repudiation. It is widely used as a measure to protect and defend information systems. This also applies to *IoT* systems, and below we explain how this concept applies to this project:

1. **Confidentiality** - Information that is stored, stayed secret unless it is intended to be revealed. We want data to be secured such that it stays private from non-

authorised parties.

2. **Integrity** - Maintaining the consistency and trustworthiness of the information over its lifecycle. We want to ensure that the blood pressure readings are trustworthy and not tampered with.

3. **Availability** - The service of making sure the offered functionality is provided when needed. We want to make sure that the user can use the blood pressure monitor when it is supposed to be available.

4. **Authentication** - The process of confirming and authenticating the user's identity. We have to make sure that the cloud system does not allow an unauthenticated user to upload readings on behalf of others.

5. **Non-repudiation** - The assurance that someone cannot deny the validity of any action. We want to make sure that the proof of origin of any action or data is protected.

Chapter 3

Methodology

In this chapter, we talk about the methods that we used to explore the vulnerabilities that are available in the device. As we mentioned before, we will be targeting the *Activ8rLives* Blood Pressure monitor in this report. We show in detail the system architecture and the communication model of the device. We then outline different attack scenarios that would apply to general blood pressure monitors. We then describe the testbed configuration that we will be using and also various tools to reverse engineer communication protocol.

3.1 *Activ8rlives* Blood Pressure Monitor

The Activ8rlives Blood Pressure Monitor is one of the most affordable models in the market and it is commonly used by households to monitor daily blood pressure. User can sync and upload measurements through smartphones. This allows the user to keep track of the blood pressure readings periodically.

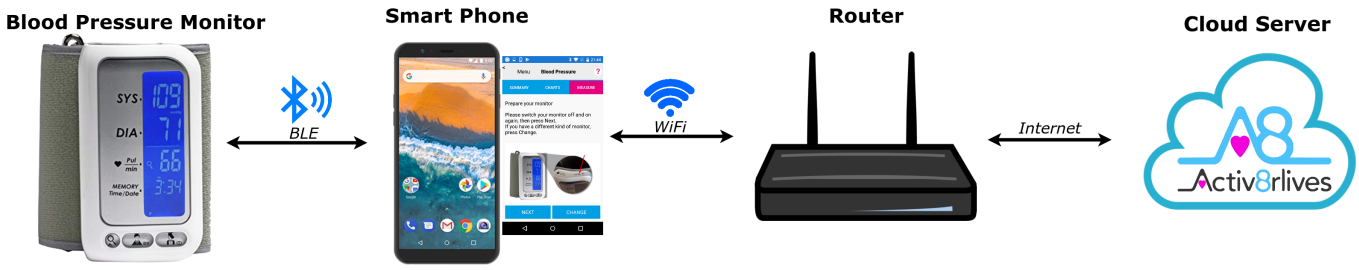


Figure 3.1: The communication model of Activ8rLives blood pressure monitor

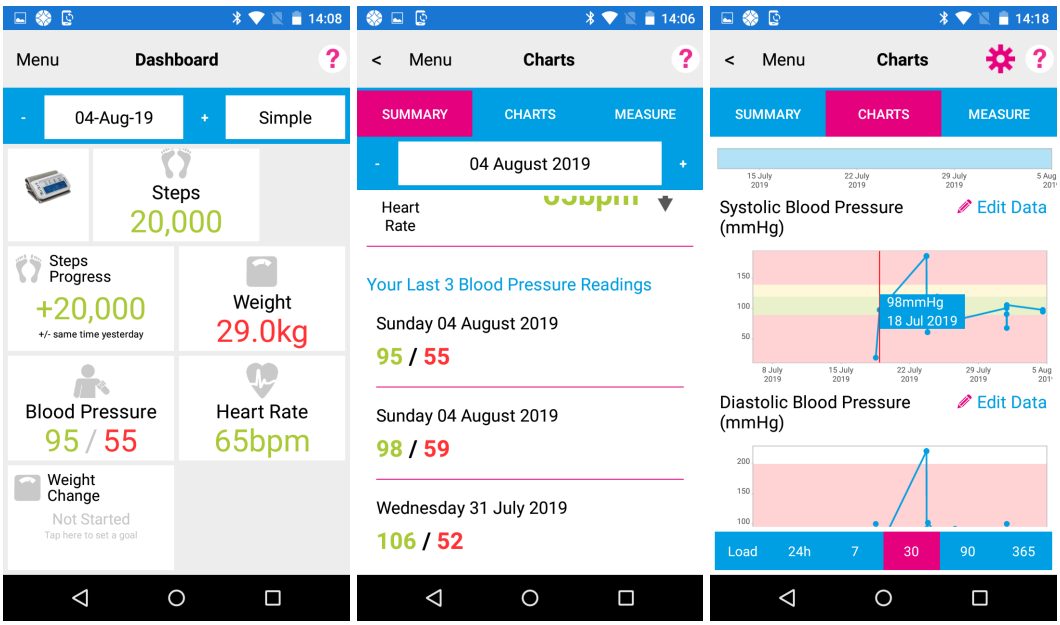


Figure 3.2: The application user interfaces.

3.1.1 Communications Model

The blood pressure monitor follows the usual IoT Paradigm. The device uses a smartphone as a gateway to upload blood pressure readings to the cloud. This is shown in **Figure 3.1**. The smartphone app is used to propagate data to the cloud server. The data is transmitted to the smartphone with BLE, and the app uploads data to the cloud using RESTful API provided by the server.

The user could read and track the blood pressure using the application. The user interface (UI) provides tracking different health information besides blood pressure readings. This is shown in **Figure 3.2**. Other than that, the user could also use the web interface dashboard to check the readings on a web browser.

Pairing

The user has to pair the device with the smartphone application before uploading blood pressure readings to the cloud server. The ”pairing” process is shown in **Figure 3.3**. The user has to switch the device into ”pairing” mode before starting the pairing process. After the ”pairing” process completes, the user does not need to re-pair the device again.

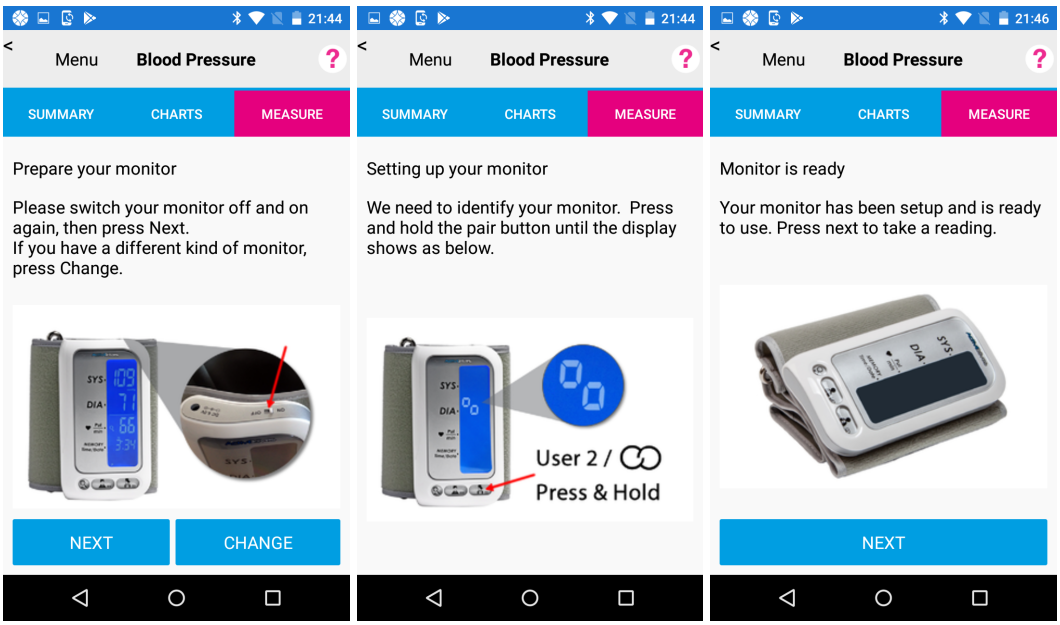


Figure 3.3: The app user interface for pairing the device with the smart phone

Reading

Blood pressure readings are uploaded right after the measurements are taken. Old measurements have to be added manually by the user. Measurements added manually are marked differently compared to measurements uploaded by the device. This shows that fake measurements added by the user could be spotted easily in the application. **Figure: 3.4** shows the reading process used by the application. The user has to trigger the reading process from the application before taking the readings.

3.2 Adversarial Model

In this chapter, we apply some threat modelling to show different attacking scenarios. We first describe the capabilities of the attacker to exploit the vulnerabilities of the device. Next, we discuss different scenarios that could benefit an attacker with nefarious intentions.

3.2.1 Attacker Capabilities

We assume that the attacker has set up a MITM proxy. This allows the attacker to transparently interfere and tamper with the communication packets.

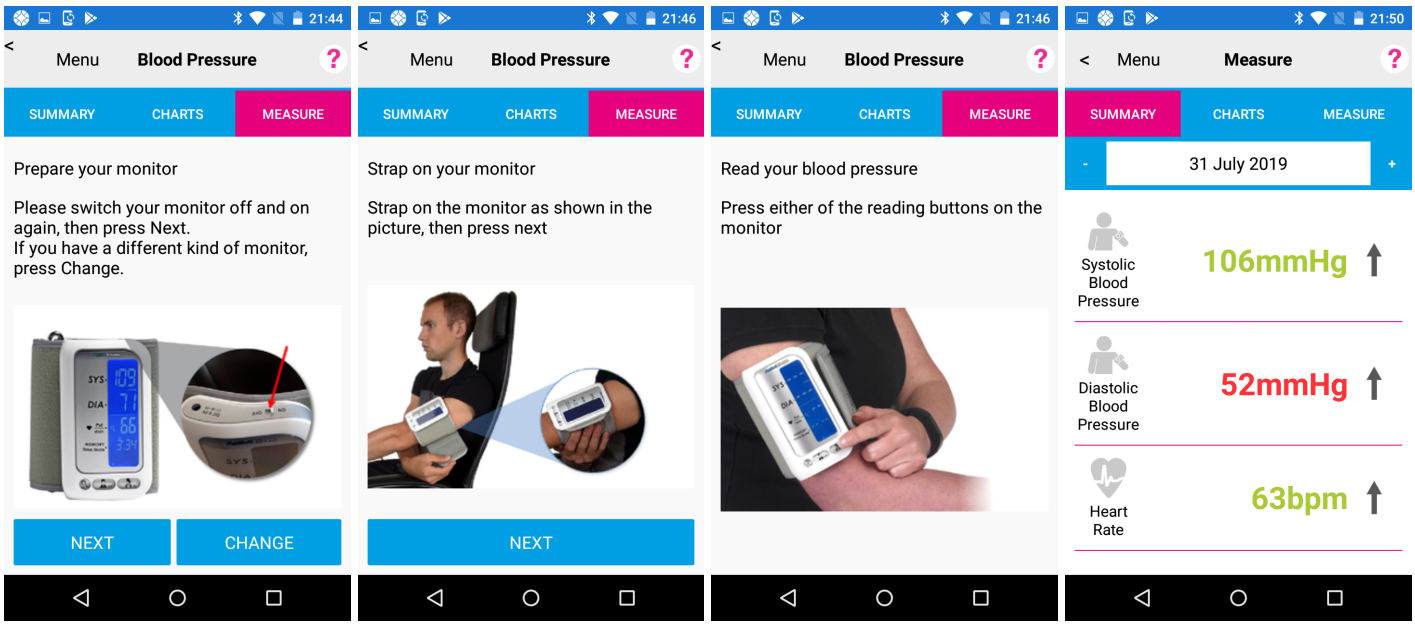


Figure 3.4: The app user interface for taking a measurements and sync to the cloud.

3.2.2 Threat Scenarios

1. **Denial of Service** : Attacker can potentially deny service by uploading a reading to the server which results overwhelms the user with a huge quantity of inaccurate information. This violates the integrity of data in the process of information assurance.

    - *Communication Hijacking* - Bluetooth communication without proper authentication might result in denial of service. The attacker can exploit this and pair with the device to hijack the communication.
    - *Fake Bluetooth Device* - An attacker could deploy smart BLE devices that could send fake measurements to the user's mobile phone.
    - *False Malware Updates* - Unsigned firmware updates could potentially allow the attacker to inject malware to either the device or the application. This would stop the device from working or, in the worst case, brick the device.

2. **User Impersonation** : User impersonation can cause harm to the user both physically and psychologically. The attacker could submit false blood pressure readings which lead to wrong medical prescriptions. This might cause a user with good health to think they have hypertension, at the same time would cause rep-

utation damage to the device manufacturer.

- *Packet Injection* - Communications that are not encrypted can be sniffed by attackers. This allows the attackers to modify and replay the packets to the server which could lead to false readings submitted to the server.
- *False Authentication* - Pairing without any authentication will allow the attacker to pair the device without the user noticing.

3. **Spying** : Spying is the act of obtaining information without the knowledge of the compromised user. The attack surface for spying can span through a wide spectrum of attack surfaces but, in this paper, we focus on the network communication between the devices.

   - *Packet Sniffing Ongoing Communications* - Data without proper encryption would expose sensitive information to attackers.
   - *Black Mailing* - The attacker could blackmail a user with sensitive health data. For example, an actor has been blackmailed on disclosing HIV positive results to the public (Post, 2015).

3.3 Testbed

We deployed a general testbed to analyse or identify any security flaws in the targeted device. The testbed was built such that we could repeat the analysis to multiple devices. We assume that the medical device uses a smartphone as a gateway to propagate data to the cloud server. We will be using a `Moto G - 3rd` with `Android` (ver 5.1.1) as our mobile device. We also used a `Linux` based laptop running `Ubuntu 18.04 LTS` as the OS to run and set up the testbed. In **Section 3.3.1**, we will discuss in detail testbed setup.

Source Code 3.1: /etc/network/interfaces

```
auto ${AP interface}
iface ${AP interface} inet static
    address 172.25.1.1
    netmask 255.255.255.0
    network 172.25.1.0
```

```
auto ${Native interface}
iface ${Native interface} inet dhcp
```

3.3.1 Setup Environment

The testbed environment is set up to allow transparent HTTP/S transmission between the phone app and the server. In the testbed, we manually set up a custom WiFi Access Point (AP) that our smartphone will be connecting to.

The *hostapd* service is used to configure the external Wifi Adapter (*ALFA Network AWUS036NHA*) as the custom Wifi AP. Meanwhile *wpa_supplicant* is used to configure the native wireless adapter to connect to an actual WiFi network. **Source Code 3.1** shows the interface configuration file. We can see that static IP address is configured for the AP interface whereas dynamic IP address is configured for the native WiFi interface.

Source Code 3.2: Port Forwarding Rules

```
iptables -A FORWARD -i ${AP interface} -o ${Native interface} -j ACCEPT
iptables -A FORWARD -i ${Native interface} -o ${AP interface} -m state
 ↪  --state ESTABLISHED,RELATED -j ACCEPT
iptables -t nat -A POSTROUTING -o ${AP interface} -j MASQUERADE
```

Besides, we used *isc-dhcp-server* to dynamically allocate IP addresses to clients that connect to the AP. At the same time, we used google DNS (8.8.8.8) with *dnsmasq* to resolve DNS names. Finally, we enabled and appended some port forwarding rules to forward the packets from the AP interface to the native interface, as shown in **Source Code 3.2**.

3.3.2 MITM Setup

We setup a *MITM* proxy on the AP interface to intercept the communication between the device and the cloud server. We simply redirect packets from port 80 (*HTTP*) and 443 (*HTTPS*) packets through port 8080 (*MITM* proxy) which is shown in **Source Code 3.3**. The final configuration of the testbed is shown in **Figure 3.5**. Note that,

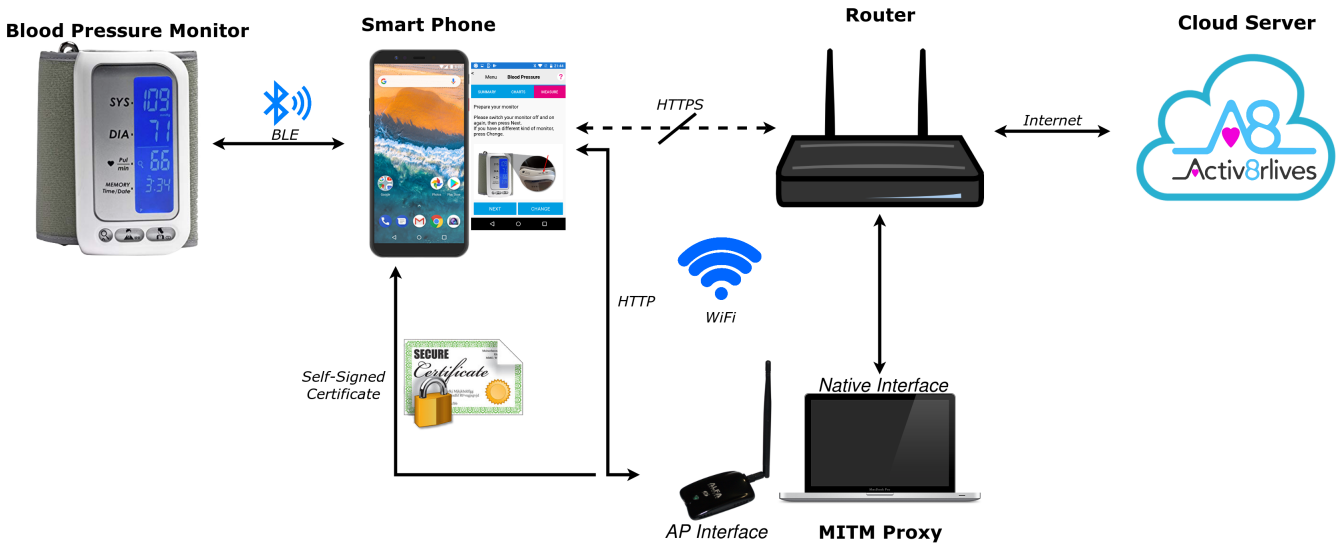


Figure 3.5: The testbed structure to setup MITM proxy.

we installed a fake Certified Authority (*CA*) certificate on the Android smartphone to allow us to look into *HTTPS* packets.

Source Code 3.3: MITM port redirect rules.

```
1 iptables -t nat -A PREROUTING -i ${AP interface} -p tcp --dport 80 -j
   ↪  REDIRECT --to-port 8080
2 iptables -t nat -A PREROUTING -i ${AP interface} -p tcp --dport 443 -j
   ↪  RERIRECT --to-port 8080
```

3.4 BLE Debugging Tools

We used various BLE debugging tools to reverse engineer the BLE protocol between the device and the app. Using these tools we can understand and analyze any security vulnerabilities available in the BLE protocol.

We used the Android app nRF Connect (Semiconductor., 2019) to understand the BLE GATT characteristics and services that are used. It provides a friendly interface to debug BLE GATT services and thus give more information on the BLE communication used by the device.

Furthermore, we logged the BLE packets between the smartphone and the device.

Logging is enabled on Android smartphones through developer mode. The logs are then analysed, using the open-source packet analyser, Wireshark (Orebaugh et al., 2007).

3.5 Application Decompiling

To further reverse engineer the protocol, we decompiled the Android application using various tools. We used APK Extractor, to extract the APK of the android app. We then further decompile the APK into source code by using APKTool (apk, 2017), Dex2JAR (pxb1988, 2018) and JDGui (java decompiler, 2019).

Chapter 4

Security Analysis

In this chapter, we talk about how we apply the exploits on the Activ8rLives Blood Pressure monitor device. In particular, we logged sensitive information using MITM proxy such as National Health Security (NSH) number. We showed that the data are structured in clear JavaScript Object Notation (JSON) format. Moreover, we injected false data into the cloud by using non-expiring tokens. Next, we reverse engineered the BLE communication protocol between the device and the application. We then programmed a BLE device to spoof fake data through the smartphone app.

4.1 Transparent Proxying

We can transparently sniff the communication between the Android application and the server. With TLS/SSL stripped from the protocol, we can fully compromise HTTPS calls to the vendor's server. This shows that the vendor used a self-signed certificate to secure communications.

This is a common vulnerability: by using MITM proxy, private information such as email, password and National Insurance Number (NHS) are being transmitted unencrypted over the air in plain JSON. JSON is a lightweight data format widely used on the web. The intercepted JSON is shown in **Listing 4.1**. This would allow the attacker to use the user's identity to gain monetary advantages.

Listing 4.1: The request body sent to the server in plain text JSON format on account creation.

```
{
    "user-type": "user",
    "user-username": "testing123@gmail.com",
```

```
    "user-password": "123456789Qwertyuiop",
    "NHS": "xxxxxxxx",
    "user-first-name": "test",
    "user-last-name": "ing",
    "user-nickname": "test",
    "user-dateofbirth": "2006-07-16T17:05:28.654Z",
    "user-gender": "M",
    "user-weight-units": "kg",
    "user-height-units": "cm",
    "user-receive-marketing": "False",
    "user-units": "metric",
    "Fev1-target": 2.25,
    "Pef-target": 283,
    "Step-target": 10000,
    "Activity-target": "60",
    "Weight-target": 66,
    "Stride-length": 60,
    "Weight": 60,
    "Height": 150
}
```


On account creation, the server generates a **token** and a **UserID**, both of which are sent as a plain text response to the user application. This is shown in **Listing 4.2**. They are also sent when the user is logging into the application as shown in **Listing 4.3**.

Listing 4.2: Profile Creation Response


```
{
  "user_id": 63378,
  "token": "a149e309c2796e11b8a467eabed7850c1aa22aee"
}
```


Listing 4.3: Login Response


```
{
  "user_id": 63363,
  "token": "06e40a343d9a183c92515bfc0fd5bb4d22f4e891",
  "password_age": "2019-07-14T19:23:39.239131+00:00"
```

```
}
```

Key	Value
Content-Type	application/json; charset=utf-8
Authorization	token e9568109372290ee92f0cecc02d0db595a64eb55
Accept-Encoding	identity
Content-Length	273
User-Agent	Dalvik/2.1.0 (Linux; U; Android 5.1.1; MotoG3 Build/LPI23.72-66)
Host	api.activ8rlives.com
Connection	Keep-Alive

Figure 4.1: HTTP Request Headers for posting measurements.

4.2 Packet Injection

Once a measurement reading is sent to the Android application through BLE, the app then sends an HTTP POST request to the server. The header and body are shown in **Figure 4.1** and **Listing 4.4** respectively. The server in return replies with a unique event id as shown in **Listing 4.5**

Listing 4.4: Post Measurements

```
{
    "guid": "5ac68840-8a76-482d-9b0a-4db60323eafe",
    "start_date": "2019-07-18T11:57:48.096448Z",
    "end_date": "2019-07-18T11:57:48.096448Z",
    "source": "bloodpressure",
    "data_types": [
      "Systolic-Blood-Pressure",
      "Diastolic-Blood-Pressure",
      "Pulse"
    ],
    "data": [0, 0, 0],
    "owner": "6636"
  }
```

Listing 4.5: Post Measurements Response

```
{
```

```
  "url": "/api/0.1/events/58601650/"
}
```

We used the MITM Proxy to intercept the communication between the app and the vendor's server. This is done while the user is taking a reading and syncing through the mobile application. The proxy intercepted the response and changed it to a different value by the attacker. The values are only tampered with in transit to the server and hence, the user would not notice any changes in either the app or the blood pressure monitor. However, we can see that the values are changed when using the web interface as shown in **Figure 4.2**.

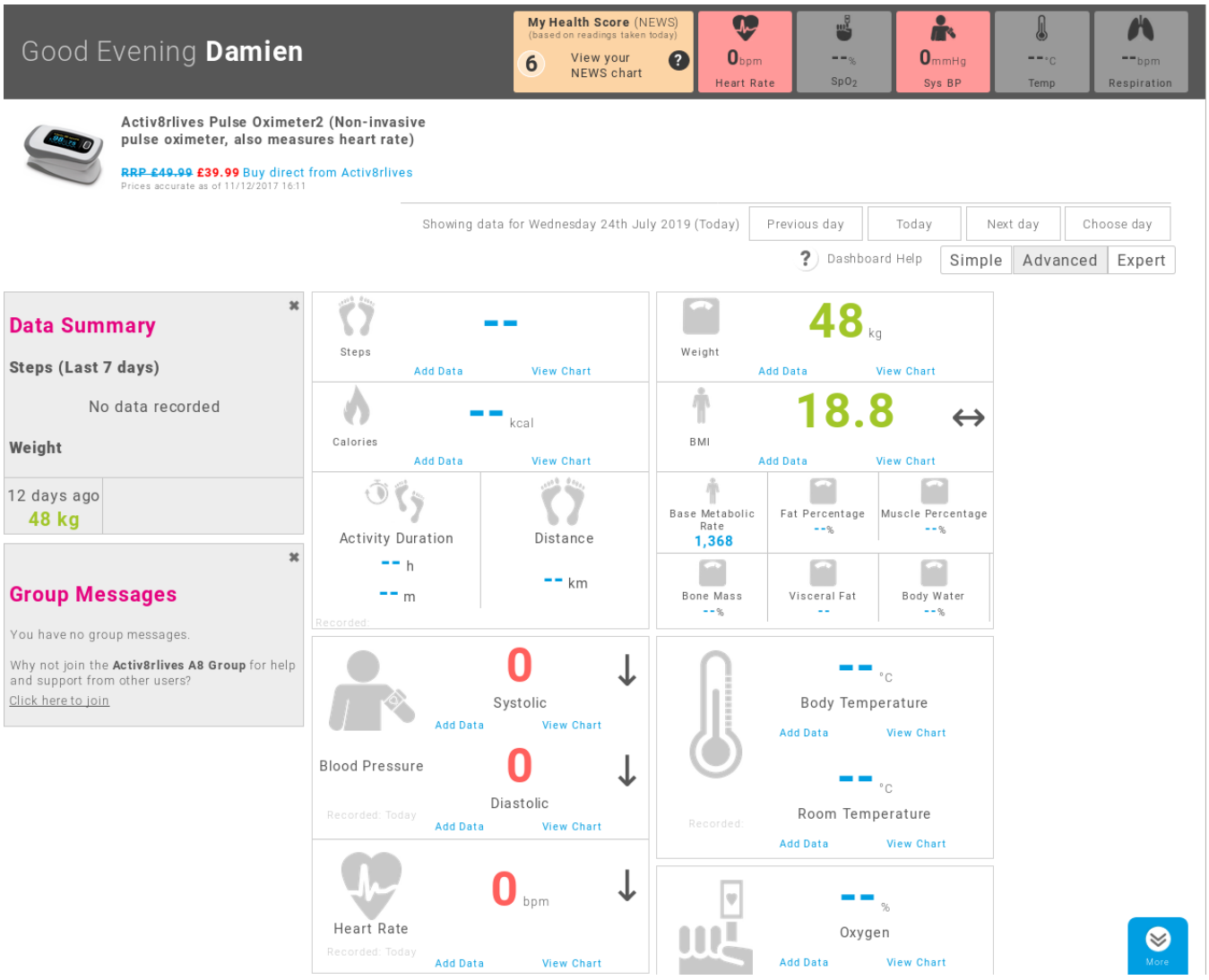


Figure 4.2: The Web Interface for Activ8rLive services with spoofed data being uploaded to the server.

We noticed that only the token and the UserID are used by the server to authenticate the user. Most importantly, the authentication token itself does not expire. This allows the attacker to apply replay attacks once the authentication token is compromised.

Source Code 4.1: get_reading.sh

```
curl -k \
-H 'Authorization:token '"${AUTHTOKEN}"  \
-H 'Host:api.activ8rlives.com' \
'https://51.179.221.21/api/0.1/ events/?owner='"${USERID}"'&
 ↪  data_types=Diastolic-Blood-Pressure &order_by=-start_date&count=1'
```

Source Code 4.2: post_reading.sh

```
curl -k \
    -H 'Content-Type:application/json; charset=utf-8' \
    -H "Authorization:token ${AUTHTOKEN}" \
    -H 'Host:api.activ8rlives.com' \
    -X POST 'https://51.179.221.21/api/0.1/events/' \
    --data-binary '[{"guid":"5ac68840-8a76-482d-9b0a-4db60323eafe",
     ↪  "start_date":"2019-07-18T11:57:48.096448Z",
     ↪  "end_date":"2019-07-18T11:57:48.096448Z", "source":"bloodpressure",
     ↪  "data_types":[ "Systolic-Blood-Pressure", "Diastolic-Blood-Pressure",
     ↪  "Pulse"],"data":[19.0,19.0,19.0],"owner"":${USERID}}]"
```

This allows the attacker to remotely retrieve or submit false readings. The attacker would only need to hijack the connection and retrieve the relevant authorization token and UserID. This could be done using a command-line tool to make HTTP calls such as *cURL*. The example of the script and response in JSON are shown in **Source Code 4.1** and **JSON 4.6** respectively. Note that we used the -k suffix in *cURL* because the vendor does not use a CA-signed certificate.

Listing 4.6: JSON response to retrieve recent measurements reading using cURL.

```
{
  "end_date": "2019-07-26T19:55:58.606800Z",
  "id": 59146118,
  "guid": "192edcf9-f17e-44d7-a832-7b56ad6e1ce6",
```

```
    "user": 63342,
    "related_to": null,
    "data": [3, 2, 1],
    "owner": 63342,
    "data_ids": [38, 53, 53],
    "data_types": ["Pulse","Diastolic-Blood-Pressure","
       Systolic-Blood-Pressure"],
    "modified": "2019-07-26T19:55:55.348649Z",
    "replaces": null,
    "url": "/api/0.1/events/59146118/",
    "start_date": "2019-07-26T19:55:58.606800Z",
    "source": "bloodpressure",
    "replaced_by": null,
    "created": "2019-07-26T19:55:55.348612Z"
  }
```


4.3 Activ8Live Application BLE Protocol

To understand the BLE protocol between the device and the Android app, we used the BLE debugger tool, Nordic Connect (Semiconductor., 2019). We can conenct with the monitor without any authentication steps required. We discovered a proprietary service with a 16-bit UUID of 0x7809 that consists of 5 different characteristics which is shown in **Table 4.1**. We can see that read1, read2 and write1 are setups with NOTIFY/INDICATE flag. This would allow the server (blood pressure monitor) to notify the client (app) if there is a new value written to the characteristic.

Now, we can infer that the device mainly uses this propriety service to communicate with the app. For us to understand the protocol, we inspect the BLE packets at a high level using HCI logging. Also, we decompiled the app to look at the source code for further clarification of the protocol.

Figure 4.3 shows the screenshot of the custom service. However, we were not able to retrieve any other information other than the serial code and firmware model from the General Access Service.

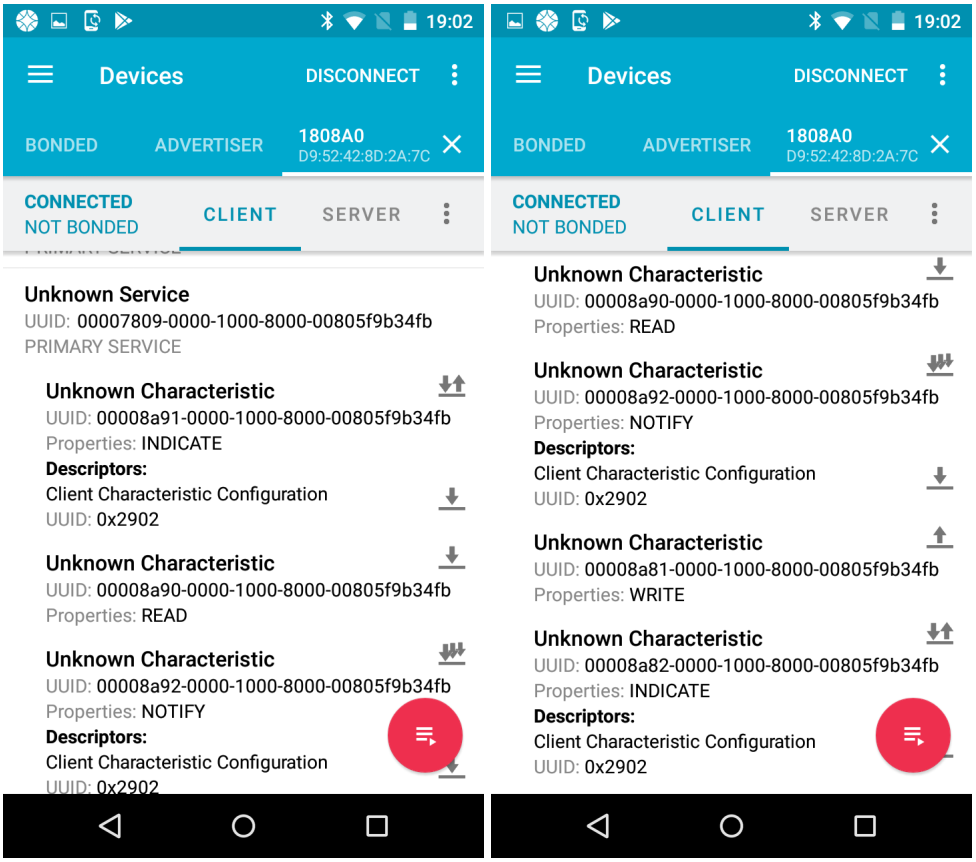


Figure 4.3: The custom service with characteristics advertised by the Activ8Live blood pressure monitor.

4.3.1 HCI Logs

In this section, we look at the BLE packets to understand and reverse engineer the communication protocol. The logs were logged and retrieved by enabling HCI logging in developer mode. **Figure 4.4** shows one of the characteristics being sent in Attribute Transfer Protocol (ATT).

We used the WireShark tool to format the data in a more readable format. The opcode with $0x1d$, show that it is an indication value sent by the GATT server. It has the handle, $0x000b$ to indicate that it is under the GATT service, $0x7809$ which contains GATT characteristic of $0x8a91$. Next, the blood pressure readings are embedded in the payload.

Source Code 4.3: Snippet of the MainActivity.java

```
1 public class MainActivity extends
    md5da774518c0af898c1651afda07f7148f.MainActivity
```

Type	UUID	NAME	FLAG
Service	0x7809	Service	READ
Characteristic	0x8A82	read1	INDICATE
	0x8A91	read2	INDICATE
	0x8A90	read3	READ
	0x8A92	write1	NOTIFY
	0x8A81	write2	WRITE

Table 4.1: The GATT proprietary service from the blood pressure monitor device as the GATT server.

```
2     implements IGCUserPeer
3 {
4
5     public MainActivity()
6     {
7         if(getClass() == com/activ8rlives/activ8rlives4/
              MainActivity)
8             TypeManager.Activate("Activ8rlives4.Droid.
                  MainActivity, Activ8rlives4.Droid", "",
                  this, new Object[0]);
9     }
10
11    private native void n_onCreate(Bundle bundle);
12 }
```

4.3.2 Application Decompiling

To further reverse engineer the communication protocol we decompiled the Android application. ApkTool, Dex2Jar and JAD were used to retrieve the source code, which is shown in **Code 4.3**. We noticed that the app was developed using the cross-platform framework, Xamarin (Devopedia., 2018). We then used a .NET decompiler (dgrunwald., 2019) to decompile the assemblies contained in the APK file.

The source code itself is in plain text and it is not obfuscated. We were able to

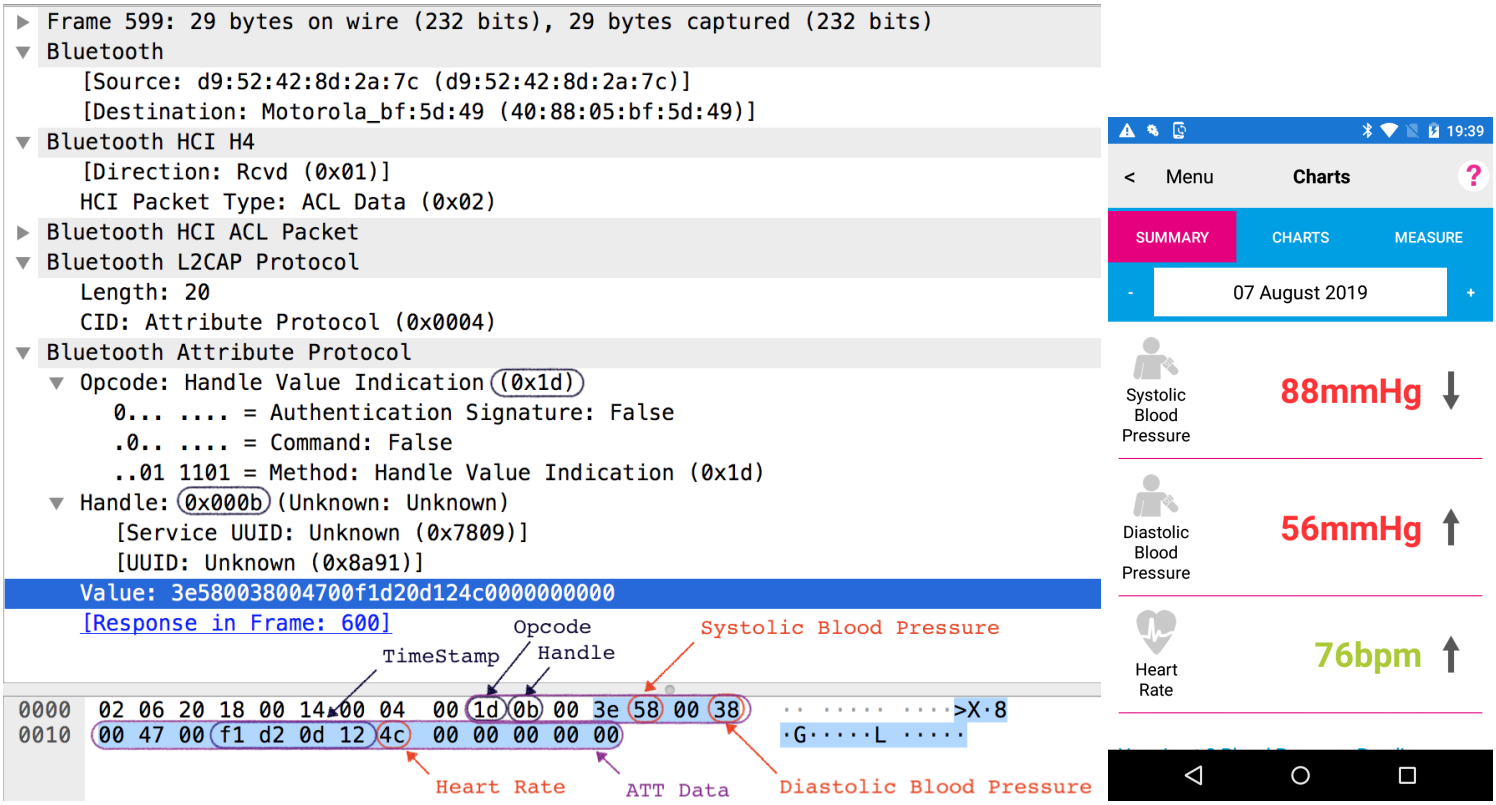


Figure 4.4: The Bluetooth HCI logs of the readings sent by the device to the app.

reverse engineer the BLE communication protocol by reading the source code. A snippet of the source code in C# is shown in **Listing 4.7** This shows a serious vulnerability, since we can understand in detail how the application was developed.

Listing 4.7: onReadPassword

```
1 public override void OnReadPassword(byte[] bytes){
2     uint num = bytes[1];
3     uint num2 = bytes[2];
4     uint num3 = bytes[3];
5     uint num4 = bytes[4];
6     uint password = num | (num2 << 8) | (num3 << 16) | (
          num4 << 24);
7     base.StateMachine.CurrentDevice = new
          BloodPressureDevice{
8         UUID = base.StateMachine.Bluetooth.
              getConnectedDevice(),
9         Password = password
```

```
10      };
11      uint num5 = (uint)(DateTime.Now - base.StateMachine.
            Epoch).TotalSeconds;
12      byte[] array = new byte[5];
13      num = (num5 & 0xFF);
14      num2 = ((num5 >> 8) & 0xFF);
15      num3 = ((num5 >> 16) & 0xFF);
16      num4 = ((num5 >> 24) & 0xFF);
17      array[0] = 33;
18      array[1] = (byte)num;
19      array[2] = (byte)num2;
20      array[3] = (byte)num3;
21      array[4] = (byte)num4;
22      base.StateMachine.Bluetooth.writeCharacteristic(
            BloodPressureContants.write2_char, array);
23  }
```

4.3.3 BLE Protocol

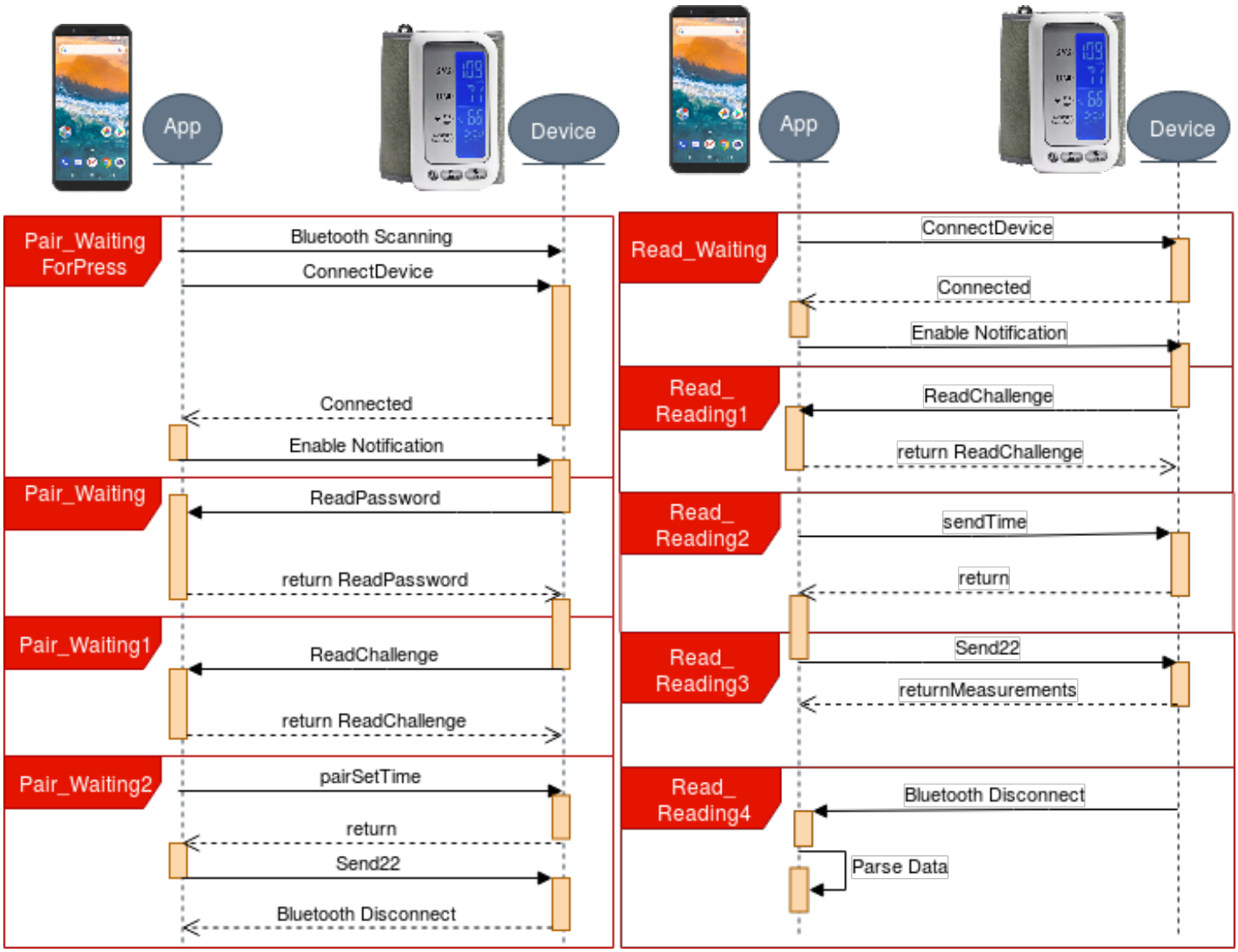
Figure 4.5 shows the BLE communication protocol used by the android application. It respectively illustrates how the device communicates with the Android application on two different use cases, *Pairing* and *Reading*. We can only infer the protocol from the application perspective since we don't have the source code of the blood pressure monitor device available.

4.3.3.1 Pairing

The application uses the pairing process to remember the UUID of the connected device. It simply uses the UUID to connect the device directly. Note that the application did not use the BLE secure pairing feature in the BLE standard. This shows that an attacker could install malware to passively logs the BLE packets.

Pairing Protocol:

1. The device has to be switched to "Pairing mode" and starts to advertise the GATT service.



(a) Pairing the device. (b) Taking a measurements.

Figure 4.5: The BLE communication protocol reverse engineered.

2. The app initiates the process by connecting for GATT services which contain characteristic UUID of $0x7809$.
3. The device initiates *ReadPassword* by sending a 5 byte ($0xA07C2A8D42$) to the phone and in return sends the current timestamp to the device. The first byte $0xA0$ is used by the app to proceed with the pairing process whereas the last 4 bytes are stored as a password associated with the UUID of the device.
4. The device then initiates *ReadChallenge* by sending another 5 bytes ($0xA103C3104D$) to the phone. Similarly, the first byte $0xA1$ is used as a flag to notify the app.

In response, the app takes the last 4 bytes of the data received and do a bit-wise XOR operation with the password received in step 2. The result is then sent to the device.

5. This follows with the app sending a $0x22$ to trigger a disconnect action to the device. The pairing process completes when the connection terminates.

Listing 4.8: The source code that shows that the application authenticates the blood pressure monitor by only using MAC address (UUID).

```
1 public override void StateStarted()
2   {
3     base.StateStarted();
4     base.StateMachine.Bluetooth.connectDevice(base.
          StateMachine.CurrentDevice.UUID,
          BloodPressureContants.svc1, null, null);
5   }
```

4.3.3.2 Reading

The application only syncs and upload measurements at the same time the user take the readings. This means that the application does not accept previously measured readings to be synced to the app. It has to be added manually by the user.

Reading Protocol:

1. The device starts to advertise the GATT service after reading was made.
2. The app searches and connects to the GATT service which contains characteristic UUID of $0x7809$.
3. The device then initiates *ReadChallenge* by sending 5 bytes ($0xA103C3104D$) to the device. The app has to take the previously stored password and apply a bit-wise XOR operation to the received challenged data. It is then sent to the device.
4. The app then sends a $0x22$ to trigger a disconnect action to the device.
5. In the meantime, the device would write the raw measurement data associated with a timestamp to the characteristic and disconnect the connection.
6. The app would then parse the data after the connection terminates.

We could infer that the monitor device checks if the application is authentic by sending *ReadChallenge* in the protocol.

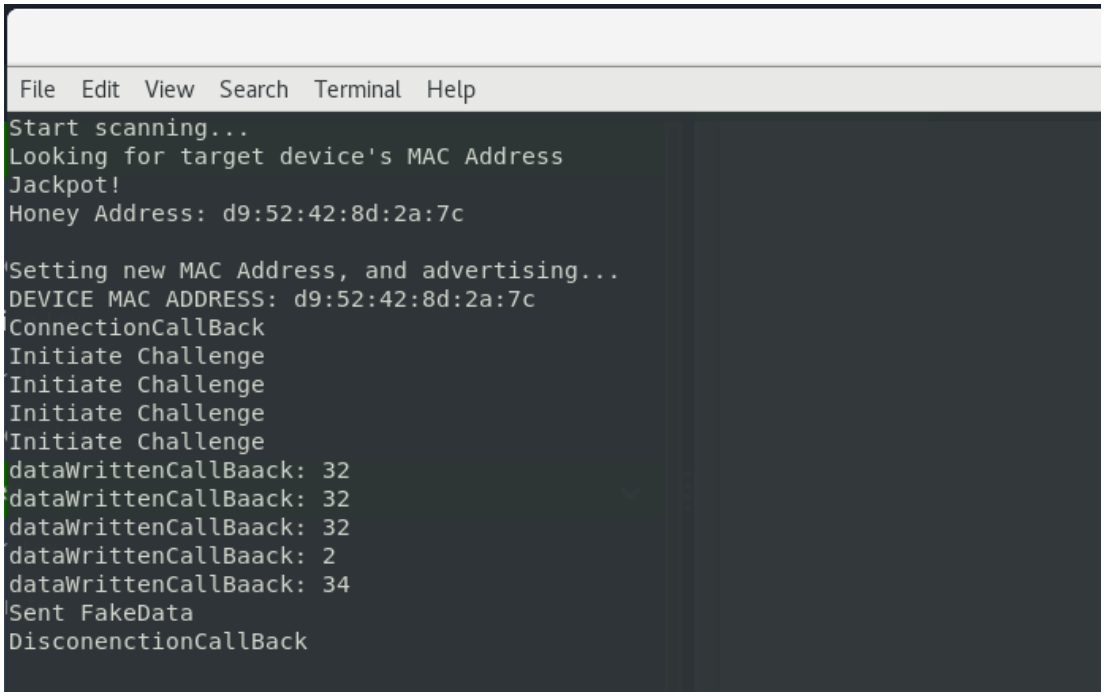


Figure 4.6: The logs of the fake BLE device in action.

4.4 Custom BLE Device

By following the protocol above, we can program a BLE device to impersonate the blood pressure monitor device. The device would be deployed in proximity to the user and try to cause a DDoS attack. This is done by spoofing fake measurements when the user is trying to sync the measurements through the application. We used a Nordic nRF51-DK development kit with the MbedOS library that provides basic BLE functionality.

The fake BLE device is programmed to work in two distinct modes. The first mode is to scan for the MAC address of the blood pressure monitor, where as the second mode is to actively spoof false measurements to the smart phone.

On startup, the fake BLE device would try to scan passively for the blood pressure monitors device. The device would look for BLE advertisement packets that contains services with UUID of $0x7809$. The device would then save the MAC address of that BLE device that advertises the packets.

Figure 4.7: The application pairs with the fake BLE device (on the right) instead of the Blood Pressure Monitor.

The fake BLE device would then actively broadcast itself with the same MAC address as the blood pressure monitor. We used the fact that the smart phone only remembers the MAC address for the 'paired' blood pressure monitor. To spoof false measurements, the fake BLE device simply follows the protocol describe in **Section 4.3.3**. **Figure 4.6** shows the logs from the fake BLE device spoofing in action.

We show that the fake BLE device can both pair and sync with the application. **Figure 4.7** shows that the application is paired with the fake BLE device instead of the blood pressure monitor. **Figure 4.8** shows spoofed data are being synchronised to the application instead of the true values.

We did a penetration testing on how likely the attack would happen. We did 20 separate measurements using the blood pressure monitor and tried to synchronise the readings to the cloud service with the fake BLE device deployed in proximity. The result is shown in **Table 4.2**. We notice that we are able to achieve 19 out of 20 spoofing for the measurements. However, this would require the fake BLE device to

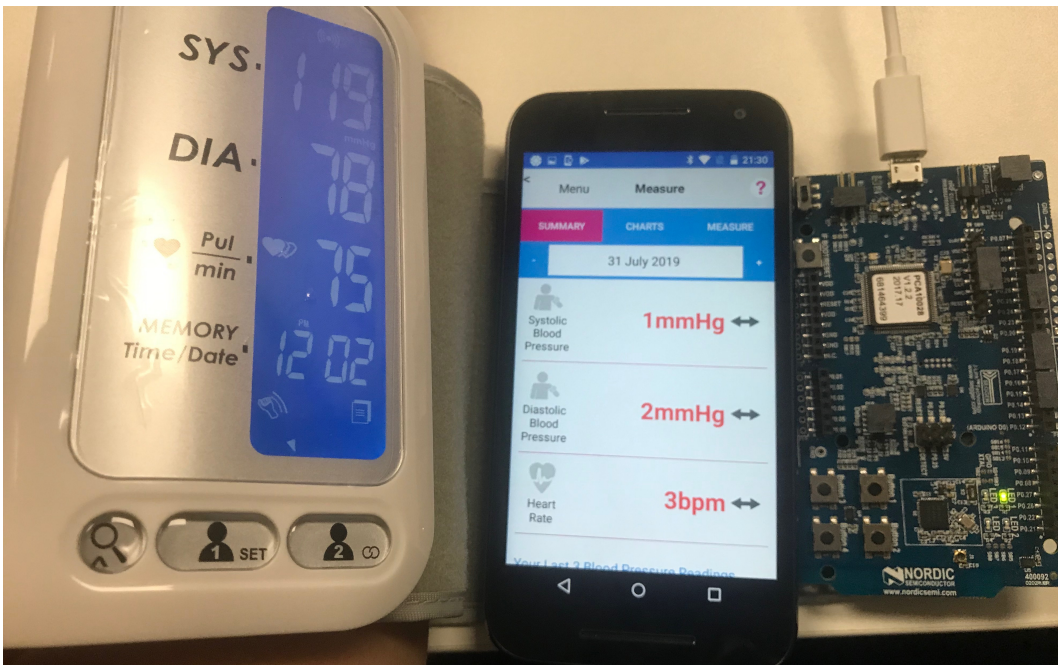


Figure 4.8: The application have received spoofed data from the fake BLE device instead of the actual reading measured with the blood pressure monitoring device.

Advertising Interval (ms)	Spoof Reading	Actual Reading
5000	0	20
1000	13	7
100	16	4
10	19	1

Table 4.2: Penetration testing with spoofing fake BLE measurements.

advertise in short advertising interval.

Here we show that it is possible to cause a DDoS attack on the mobile application. However, this attack will work only if the fake BLE device is being selected to be paired during the pairing stage. This is because the application remembers the device's BLE UUID that it pairs.

4.5 Denial of Service

By using another smartphone with Bluetooth capabilities, we can deny the connection between the user's application and the blood pressure monitoring device. This is shown

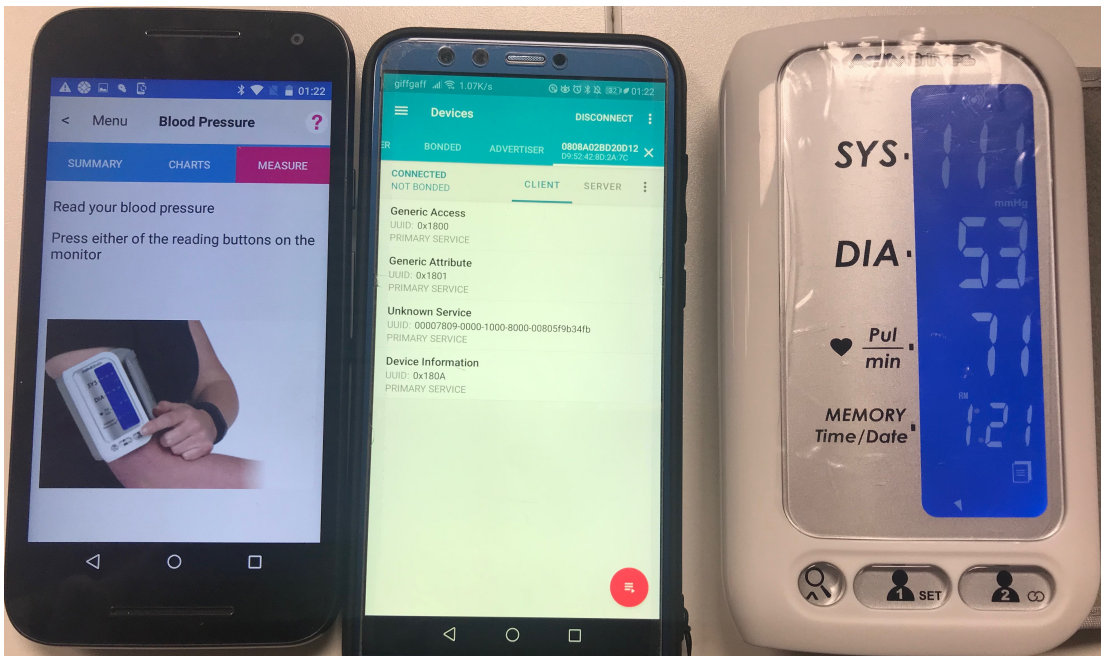


Figure 4.9: Another BLE device that connects to the blood pressure monitor without any authentication to deny connection with the application.

in **Figure 4.9**.

We have a second smartphone set up to scan and connect to the blood pressure monitor when it is broadcasting. We can stop any other phone to connect to the monitor. This includes the user's phone that was paired to the monitor.

This is possible because there are not any authentication process connecting the blood pressure device. The connection itself does not include the process of pairing or bonding. We show that this exploit could be done by an attacker with any smartphone with BLE capabilities.

Chapter 5

Discussion

In this chapter,we provide several recommendations to the design implementation to mitigate these attacks. These recommendations can also be applied to other smart blood pressure monitors or more general, *IoT* devices.

5.1 Attack Mitigation

Certificate Pinning

We can tamper with the network packets by using MITM proxy on the android smartphone. This is possible due to the trust of the developer with the client's device. Usually, the user can be easily tricked into installing a self-signed certificate into the device to allow MITM attacks to succeed (Park et al., 2020). One way to mitigate this is to implement certificate pinning in the application.

Certificate pinning is the process of associating the host with the expected certificate. This is usually done at development stage such that the certificate is stored in the application. Other than that, the certificate could also be pinned on the first connection to the server. Hence, if the connected server does not provide the pinned certificate, the connection is then unsafe and should be dropped.

In general, there are various ways to bypass certificate pinning available (Daz-Snchez et al., 2019). However, with certificate pinning, the complexity of the technique to bypass it is increased.

Token Refreshing

Token refreshing should be used to prevent impersonation attacks. The authorization token used by the application does not expire, hence, the token is associated with an account for a lifetime. An attacker would only need to compromise the authorization token once and as a consequence, the attacker would be able to apply client replay attacks remotely.

To circumvent that, the token should be only be valid for a certain timeframe. Authentication should be made after the token is expired, to request a new token. Therefore, token compromised by an attacker would only be valid for a certain timeframe.

However, this introduces bad user experience to the mobile application. This is because the user would need to enter login credentials every time the token is expired. This could be improved by adding the notion of refresh token and access token. A refresh token is a token that never expires whereas the access token is short-lived. The access token is used by the clients to authenticate themselves whereas the refresh token is used to request new access tokens when the old access token is expired.

One would ask, what happens when the refresh token itself is compromised. The attacker would then be able to request for authorization token and this destroys the purpose of having refreshing tokens. This is not the case, the server would be able to detect if the refresh token is compromised. This can be done because the attacker and the user would invalidate each other when they are trying to request a new access token, which can be detected by the server.

Code Obfuscation

Code obfuscation should be applied to the source code on production. Code obfuscation is the process of making source code unreadable, while maintaining the same functionality. This is important because source code can be reverse engineered easily with the help of decompilers available online. In particular, the source code of *android* application can be decompiled and recovered fairly easily (Eddy, 2014).

There are various tools to obfuscate source code such as Xamarin Dotfuscator (PreEmptive, 2014). It is a NET. obfuscator that obfuscates the code and is developed using the Xamarin cross-platform framework. This would make the application harder to reverse engineer by the attacker. In general, code obfuscation should be used in production. However, code obfuscator does not prevent the application to be reverse engineered, thus it is still important to have other security measures implemented in

the production.

Encryption of Sensitive Data

Important or sensitive data should be encrypted before transmission. This is because the attacker would be able to sniff and log sensitive information, i.e., we demonstrated that by using MITM Proxy, we can log account, password or NHS number. The sensitive data should be encrypted with an extra layer of encryption to ensure that data cannot be easily retrieved easily by an attacker.

BLE Security Authentication

BLE's security feature should be used for pairing. This should be implemented because un-authorised pairings could result in different attacks. In this paper, we demonstrated that another user with a BLE device could pair with the blood pressure monitor without any authorization. As a result, this causes the device to be denied service.

Other than that, BLE security pairing also provides end-to-end encryption. This would prevent sensitive information to be sniffed by an eavesdropper.

5.2 Future Work

Future work in this paper concerns a more thorough security analysis on different attack vectors. This would include applying different common exploits on targeted *IoT* devices.

The top security vulnerabilities for *IoT* in *OWASP* consists of hardware vulnerabilities (Fredric Paul, 2019). This includes hardware tear-down of blood pressure monitors to be able to extract information from left out debugging interfaces. Thus, we left this hardware security analysis as a potential future work.

Besides, the secured update mechanism is often not implemented in both the firmware of the devices and the application. This could lead to other exploits, such as injecting malware into the firmware to apply a DDoS attack. This investigation could be done on the targeted device as future work

Other than that, a comparative security study could be done on blood pressure monitoring devices from different vendors. Hence we can have a broader and more general security analysis on these devices. Other than that, we can show that what are the common security vulnerabilities that exist in the device.

Finally, with all these security vulnerabilities in mind; research on ways to detect and mitigate vulnerabilities on *IoT* devices should be done as future work. Oza et al. (2019) shows a framework of preventing MITM attacks by luring attackers using decoys. Other than that, Duan et al. (2019) proposed a security framework for published/subscribe based *IoT* communication.

Chapter 6

Conclusions

Security analyses that we conducted in this work show that *IoT* devices, more specifically, health monitoring devices, contain serious security flaws. We applied multiple security analyses on the Activ8rLives Blood Pressure Monitor that span through multiple attack vectors.

This includes communication hijacking using MITM proxy, compromising authentication token to do client replay attacks, application decompiling with decompiling tools, reverse engineer the BLE protocol and BLE device spoofing.

These are all general and common vulnerabilities and well known in the industry. However, these vulnerabilities are not prevented and considered to be important in production by product designers. This results in security exploits that are expensive to patch by manufacturers. Moreover, these security patches and updates are not enforced by the manufacturers. Other than that, the user does not upgrade their firmware as much as they should. These lead to *IoT* devices to contain vulnerability indefinitely.

Bibliography

A tool for reverse engineering 3rd party, closed, binary android apps. https://ibotpeaches.github.io/Apktool/, 2017.

Accenture. Accenture. digital trust in the iot era, 2015. URL https://www.accenture.com/t20150714T123236__w__/_acnmedia/Accenture/Conversion-Assets/DotCom/Documents/Global/PDF/Dualpub_18/Accenture-Digital-Trust.pdf.

Classen, Jiska, Wegemer, Daniel, Patras, Paul, Spink, Tom, and Hollick, Matthias. Anatomy of a vulnerable fitness tracking system: Dissecting the fitbit cloud, app, and firmware. *Proceedings of the ACM Interactactive and Mobile Wearable Ubiquitous Technology*, 2(1):5:1–5:24, March 2018.

Cyr, Bruce D., Horn, Webb, Miao, Daniela, and Specter, Michael A. Security analysis of wearable fitness devices (fitbit). 2014.

Davidson, Robert. Getting started with bluetooth low energy. 2014.

Devopedia. Xamarin, version 11, 2018. URL https://devopedia.org/xamarin.

dgrunwald. Ilspy, open-source .net assembly browser and decompiler, 2019. URL https://github.com/icsharpcode/ILSpy.

DigitalHealth. Special report: Remote monitoring and self-care, 2018. URL https://www.digitalhealth.net/2018/04/special-report-remote-monitoring-and-self-care/.

Duan, L., Sun, C., Zhang, Y., Ni, W., and Chen, J. A comprehensive security framework for publish/subscribe-based iot services communication. *IEEE Access*, 7: 25989–26001, 2019. ISSN 2169-3536. doi: 10.1109/ACCESS.2019.2899076.

Daz-Snchez, D., Marn-Lopez, A., Almenarez, F., Arias, P., and Sherratt, R. S. Tls/pki challenges and certificate pinning techniques for iot and m2m secure communications. *IEEE Communications Surveys Tutorials*, pp. 1–1, 2019. ISSN 1553-877X. doi: 10.1109/COMST.2019.2914453.

Eddy, Max. Rsac: Reverse-engineering an android app in five minutes, 2014. URL https://uk.pcmag.com/opinion/10593/rsac-reverse-engineering-an-android-app-in-five-minutes.

Fereidooni, H., Classen, J., Spink, T., Patras, P., Miettinen, M., Sadeghi, A.-R., Hollick, M., and Conti, M. Breaking fitness records without moving: Reverse engineering and spoofing Fitbit. In *Proceeding of International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, Atlanta, GA, USA, September 2017.

Fielding, Roy, Gettys, Jim, Mogul, Jeffrey, Frystyk, Henrik, Masinter, Larry, Leach, Paul, and Berners-Lee, Tim. Hypertext transfer protocol–http/1.1, 1999.

Fredric Paul, Networkworld. Top 10 iot vulnerabilities, 2019. URL https://www.networkworld.com/article/3332032/top-10-iot-vulnerabilities.html.

Google. Secure your site with https, 2019. URL https://support.google.com/webmasters/answer/6073543?hl=en.

IoT_Agenda. Current and future applications of iot in healthcare, 2018. URL https://internetofthingsagenda.techtarget.com/feature/Can-we-expect-the-Internet-of-Things-in-healthcare.

iscoop.eu. Iot security: smart business requires smarter internet of things security, 2018. URL https://www.i-scoop.eu/iot-security-smarter-internet-of-things-security/.

java decompiler. Java decompiler, yet another fast java decompiler., 2019. URL https://java-decompiler.github.io.

Jeon, K. E., She, J., Soonsawad, P., and Ng, P. C. Ble beacons for internet of things applications: Survey, challenges, and opportunities. *IEEE Internet of Things Journal*, 5(2):811–828, April 2018. ISSN 2327-4662. doi: 10.1109/JIOT.2017.2788449.

Maher, Carol, Ryan, Jillian, Ambrosi, Christina, and Edney, Sarah. Users' experiences of wearable activity trackers: a cross-sectional study.(survey). *BMC Public Health*, 17(1), 2017. ISSN 1471-2458.

Matt Toomey, Aberdeen. Iot device security is being seriously neglected, 2018. URL https://www.aberdeen.com/techpro-essentials/iot-device-security-seriously-neglected/.

Orebaugh, Angela, Ramirez, Gilbert, Beale, Jay, and Wright, Joshua. *Wireshark & Ethereal Network Protocol Analyzer Toolkit*. Syngress Publishing, 2007. ISBN 1597490733, 9781597490733.

Orlosky, Jason, Ezenwoye, Onyeka, Yates, Heather, and Besenyi, Gina. A look at the security and privacy of fitbit as a health activity tracker. In *Proceedings of the 2019 ACM Southeast Conference*, pp. 241–244. ACM, 2019.

Oza, Antara Durgesh, Kumar, Gardas Naresh, Khorajiya, Moin, and Tiwari, Vineeta. Snaring cyber attacks on iot devices with honeynet. In Peng, Sheng-Lung, Dey, Nilanjan, and Bundele, Mahesh (eds.), *Computing and Network Sustainability*, pp. 1–12, Singapore, 2019. Springer Singapore. ISBN 978-981-13-7150-9.

Park, Junghoon, Son, Byeonggeun, Park, Junyoung, Kim, Myoungsu, and Yim, Kangbin. Unintended certificate installation into remote iot nodes. In Barolli, Leonard, Xhafa, Fatos, and Hussain, Omar K. (eds.), *Innovative Mobile and Internet Services in Ubiquitous Computing*, pp. 845–854, Cham, 2020. Springer International Publishing. ISBN 978-3-030-22263-5.

Post, Washington. Charlie sheens hiv status and the dawn of medical-data blackmail, 2015. URL https://www.washingtonpost.com/news/to-your-health/wp/2015/11/17/charlie-sheens-hiv-status-and-the-dawn-of-medical-data-blackmail/?noredirect=on.

PreEmptive. Protecting xamarin apps, 2014. URL https://www.preemptive.com/dotfuscator/pro/userguide/en/getting_started_xamarin.html.

pxb1988, Bob Pan. dex2jar, decompile dalvik executable (.dex/.odex) format., 2018. URL https://github.com/pxb1988/dex2jar.

ReadWrite. The 5 ways iot is about to change healthcare as we know it, 2018. URL https://readwrite.com/2018/05/22/the-5-ways-iot-is-about-to-change-healthcare-as-we-know-it/.

Scotland, Gov. Heart disease in scotland, 2019. URL https://www2.gov.scot/Topics/Health/Services/Long-Term-Conditions/Heart-Disease.

Scott Amyx, TechBeacon. 67 open source tools and resources for iot, 2018. URL https://techbeacon.com/app-dev-testing/67-open-source-tools-resources-iot.

Semiconductor., Nordic. nrf connect sdk, the bluetooth software development kit, 2019. URL https://www.nordicsemi.com/Software-and-Tools/Software.

Service, Indo Asian News. Cyberattacks grew 22% on india's iot deployments in q2, 2019. URL https://www.livemint.com/technology/tech-news/cyberattacks-grew-22-on-india-s-iot-deployments-in-q2-1565356789868.html.

SIG., Bluetooth. Bluetooth core specification v5.0, 2019. URL https://www.bluetooth.org/DocMan/handlers/DownloadDoc.ashx?doc_id=421043.

Symantec. Mirai: what you need to know about the botnet behind recent major ddos attacks, 2016. URL https://www.symantec.com/connect/blogs/mirai-what-you-need-know-about-botnet-behind-recent-major-ddos-attacks.

TheGuardian. Hackable implanted medical devices could cause deaths, 2019. URL https://www.theguardian.com/technology/2018/aug/09/implanted-medical-devices-hacking-risks-medtronic.

Tuptuk, Nilufer and Hailes, Stephen. Security of smart manufacturing systems. *Journal of Manufacturing Systems*, 47:93 – 106, 2018. ISSN 0278-6125. doi: https://doi.org/10.1016/j.jmsy.2018.04.007. URL http://www.sciencedirect.com/science/article/pii/S0278612518300463.

WPP. World population prospects, key findings and advance table, 2017. URL https://esa.un.org/unpd/wpp/Publications/Files/WPP2017_KeyFindings.pdf.

Yaqoob, T., Abbas, H., and Atiquzzaman, M. Security vulnerabilities, attacks, countermeasures, and regulations of networked medical devices a review. *IEEE Communications Surveys Tutorials*, pp. 1–1, 2019. ISSN 1553-877X. doi: 10.1109/COMST.2019.2914094.