Security Analysis of IoT Fitness devices with ARM Architecture

by

Shreyas Shetty

May 2020

Table of Contents

Abstract

We live in a world where many daily aspects of humans involve using connected devices. This may be paying for your take-outs without using your physical cards, controlling your home lights using mobile phones, setting the temperature of the home without physically being present, and many such things. These devices come under the umbrella of IoT or Internet-of-Things or to put it simple terms, adding Internet connections to your devices.

Smart devices are very common nowadays, where the term ‘Smartness’ directly relies on the ‘connectivity’ of these devices. IoT helps devices to be controlled from anywhere with an Internet connection and gather information remotely across the ecosystem where many other devices are connected and registered under a single user. This also helps in reducing effort to work multiple devices to perform a task. There are also cheaper variants available in the market which attracts users in a large scale. The cheaper variants fall light on the budget, but have many loopholes which are later discovered and been exploited which cause harm to customer’s confidentiality and cost the service providing company a dip in its revenue.

The goal of this project is to evaluate the vulnerabilities present in the smart devices provided by one of the largest producers of smart wearables and smart home devices. Then we create a prototype to exploit them in terms of viewing personal information of the victim user. When it comes to the cost of buying these devices, it comes at a very reasonable price from its competitors but when it comes to security and confidentiality of devices, it can be very easily compromised. In the end, we also propose some tangible suggestions for improving the security of these low-budget devices.

Keywords

IoT (Internet-Of-Things), BLE (Bluetooth Low Energy), IoT Cloud, APK (Android Package Kit), ARM Architecture, IoT Firmware.

1. Introduction

The smart device [22] is termed to one that incorporates advanced sensing through sensors in the device and stores the data within it and transfers to the application within the main device like a smartphone for alerts and further actions. (E.g., text message notifications, call notifications, heart rate, calories burned, number of steps walked, low battery alerts, GPS notifications). These events help in saving time and effort of users in reaching out for smartphones to view important information and access grants. (E.g., view text message and notifications on smart watches).

In these past years, Smart wearables are more easily eye-catching these days on almost every hand you see. They are not just a medium for tracking fitness goals like its predecessors. Take for example, the NFC [18] technology introduced in Mi Band 3 which does payments at stores and metro scanners without the physical bank card. According to VXCHNGE, there will be 41 Billion IoT devices by 2027. To put this in simple terms, every second there are 127 devices connecting to the Internet. They are in the form of watches, lockets, small trackers fit in shoes, etc. It started off as fitness [19] devices, but now with the modern advancements, we can do so much more with the device other than tracking your heartbeat.

Figure 1. Smart Wearables available in the market

The size of IoT devices are compact and easy to wear or carry, but the logic and design behind its construction is very complex and has evolved through advancements in technology. (E.g., In 1998, Seiko released Ruputer which was a 2-inch screen smartwatch). Twenty years later, the number of IoT nodes attaching itself to the global information grid but has also increased the number of security issues each year. Also, each IoT vendor having their own production knowledge is creating a playground for attackers to choose whom to attack.

Through the above observations, we try to play the role of an attacker by closely examining and evaluating the security [22] of Xiaomi smart bands. Xiaomi [1] outside of China is mostly known for manufacturing smartphones, but then they quickly expanded their range to Internet-of-Things devices. According to Xiaomi as of 2018, the number of connected devices excluding smartphones and laptops on Xiaomi's [15] IoT platform is 150.9 million. We will first study the device behavior [2] and understand that BLE devices [3] have limited services and each service has some characteristics. Device behavior comprises of how data travels to and from the device and mobile application. Through this we would look out for loopholes that could be exploited and severity of the vulnerability. We found that some of the characteristics have descriptors. Some of the characteristics have only read/write access and some are more complicated to work on. To demonstrate these vulnerabilities, we developed an attacking prototype, in which we successfully control the Mi Band 3 [5] via Linux. After this, send some unexpected phone call notification and track hear-rate. Since there devices collect user data, user privacy is greatly endangered.

For the remainder of this project, we would be to reverse engineer the factory Firmware [16] that causes the vulnerabilities and try to patch each of it. The root cause for our success exploits is that almost all the time it is discoverable, work on the same frequency and allows any devices to connect to it. We propose cryptographically signing [6] the Firmware [24] as a defense mechanism. Lastly, we would conclude by displaying the importance of having a secure and cryptographically signed Firmware [29].

2. Background

2.1. Internet-of-Things (IoT)

The Internet-of-Things, or IoT, can be referred to the billions of physical devices around the world that are now connected to the internet, sensing, collecting and sharing data. With the availability of super-cheap computer chips, processors and the availability of wireless or cellular networks, we can communicate, from something as small as a capsule (E.g. Dog tracking device) to something as big as a mini airplane (E.g. Army drones for surveillance), into a world of IoT. Connecting all these different objects and adding sensors to them adds a level of digital intelligence to devices that would be otherwise dumb, enabling them to communicate real-time data without involving a human being. The Internet of Things [25] is making the fabric of the world around us smarter and more responsive, merging the digital and physical universes.

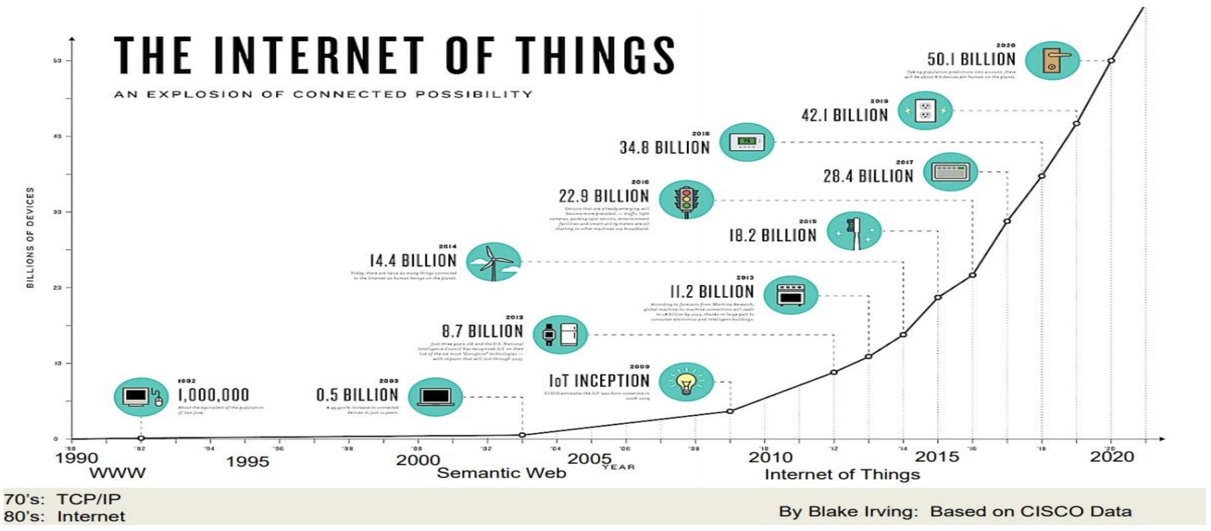


Figure 2. IoT Device users' graph

2.2. IoT Platform Architecture

IoT platforms are divided into categories as per their usage namely, commercial, corporate, farming and even in industries. They are all the same when it comes to the design aspect of the IoT device. The logic changes depending upon the business model the device was built for. From the Figure below we can see that the IoT Cloud [13] is the brains of the whole architecture. We can compare this to a Server which holds all essential information (E.g. Username, Password, Personal Details) and functions (E.g. Authentication, Analysis Graph, Alerts). When we buy a new IoT device the first thing the device and user go through is Authentication, so that accessibility is maintained throughout the lifetime of the device. Once the authentication is done, the user can use the features of the IoT device via the mobile application (E.g. Turn on home lights from your mobile). Hence, we can say that the IoT Cloud is the middleware component which handles authentication and event handling between IoT devices and mobile application.

IoT devices comprise of sensors, trackers and other sensing hardware which can sense the environment and execute appropriate events or notifications. According to the interaction of the device with the IoT Cloud, the devices are classified into Cloud-connected (E.g. Smart bulb interacts via the Cloud) or Hub connected (E.g. Door sensors interacts with Smart hub/gateway) device. There is another set of IoT devices that don't interact with the Cloud [12] or a hub but interacts directly with the IoT application on a smartphone (E.g. Mi band interacts with Mi Fit application). Such kind of devices make use of a special type of Bluetooth called BLE (Bluetooth Low Energy) for interaction purposes and exchanges data when synchronized with the mobile application.

The last component [7] of the architecture is the Mobile Application. These are the applications that help in installation (E.g. Setup Smart home ecosystem) and synchronizing (E.g. Binding IoT device with App account) the IoT device to your smartphone or laptop. It also provides an interface to view reports about the device (E.g. Power consumption report for a Smart Bulb). The mobile applications are in direct connection with the IoT Cloud [13] which shares Device Identification, Authentication [11] and Event Exchange.

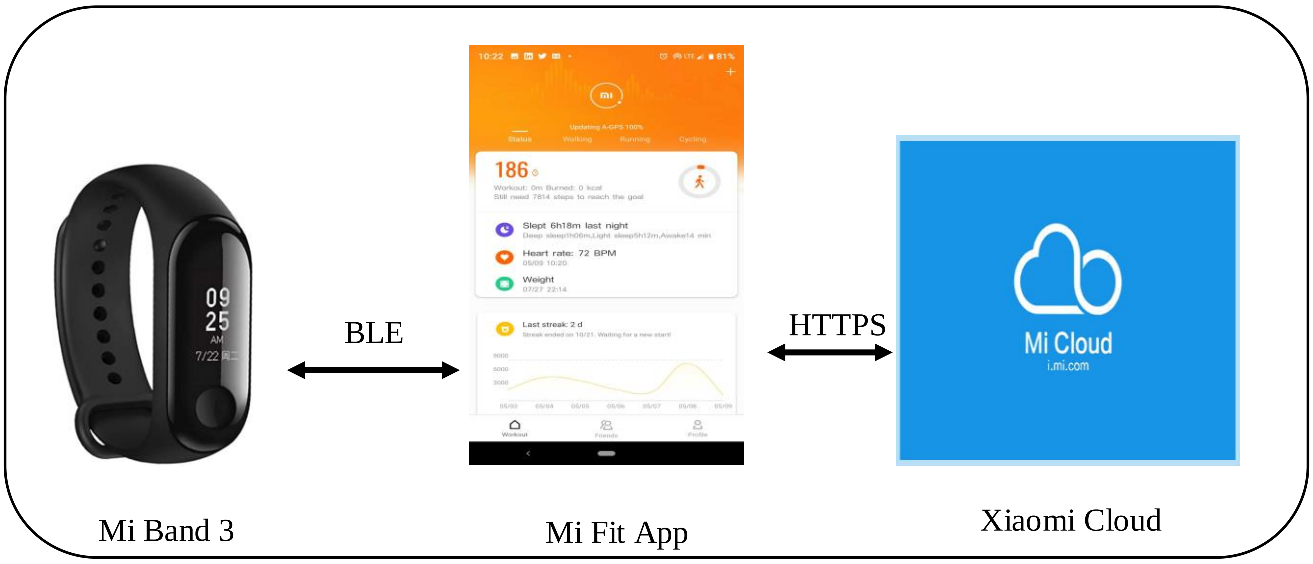


Figure 3. Xiaomi Ecosystem

2.3. Communication Model

Here we talk about the different ways the IoT device communicates with the IoT Cloud [13] and mobile application. There are four ways an IoT device could interact with the Cloud and app for the purpose of controlling the device or for data transmission.

2.3.1. Remote Control

This is like the general remote control; we have at home (E.g. Remote control for all the air condition system). So, looking from the IoT perspective we can use our smartphone (Remote Control) which has the Mobile application can be used to control the IoT device (E.g. Turn on the smart bulb away from home using Mobile app). There can be a time where you might have lost your IoT device like Smart Watch, we can use the Mobile app to track the last location of the device. We can also send alerts to the device like vibrate when near.

2.3.2. Local Control

This method of communication works when the IoT device and device controlling it are in the same network or home network (E.g. You are at home and turn on smart bulb). The mode of controlling IoT devices in a Local Control domain does not require the intervention of the IoT Cloud [12]. Through this we can see that IoT architecture is smart enough to identify whether function requests are coming from a local connection or a remote connection.

2.3.3. Smart Control

This is the control mechanism that separates IoT devices from regular electronic devices (E.g. Amazon Echo vs JBL Bluetooth [39] Speakers). Smart control makes IoT devices work by themselves without the any human intervention. For this the user feeds pre-conditions into the IoT device to which the device does the respective function. (E.g. If the room temperature goes above 65^0F, “Turn On” air conditioner). Thus, for this the IoT device senses the surrounding area to check pre-conditions are still existing. If a function needs to be performed, the IoT device interacts with the IoT Cloud for making decisions and choosing the appropriate function.

2.3.4. Data Uploading

Whenever an IoT device senses it surrounding it constantly keeps recording and sends the data [4] to the IoT Cloud. Apart from this, the IoT device also send three important messages i.e. a command response, heart-beat message and event notification. A command response is from the IoT device to the IoT Cloud to assure the Cloud that the command has been successfully executed (E.g. Turn On air condition = Success). Next comes the heart-beat message which is like a connection alive signal sent from the IoT device. This message is sent to the IoT Cloud to make it aware that the device is still present. The last message is event notification which occurs when an IoT device senses presence within its surroundings and information about this is sent to the IoT Cloud (E.g. Heart rate over 90 bps send alert)

Now we know the mediums through which an IoT device can communicate. In this project, the Mi Band 3 uses BLE [3] (Bluetooth Low Energy) for communication amongst each other. BLE is the type of Bluetooth which is in devices that do not need continuous communication with its parent device (Example: Smartwatch and Mobile Application inside a Smartphone). Hence, from the name, it suggests it requires low energy to function correctly making it more economical to use after purchase.

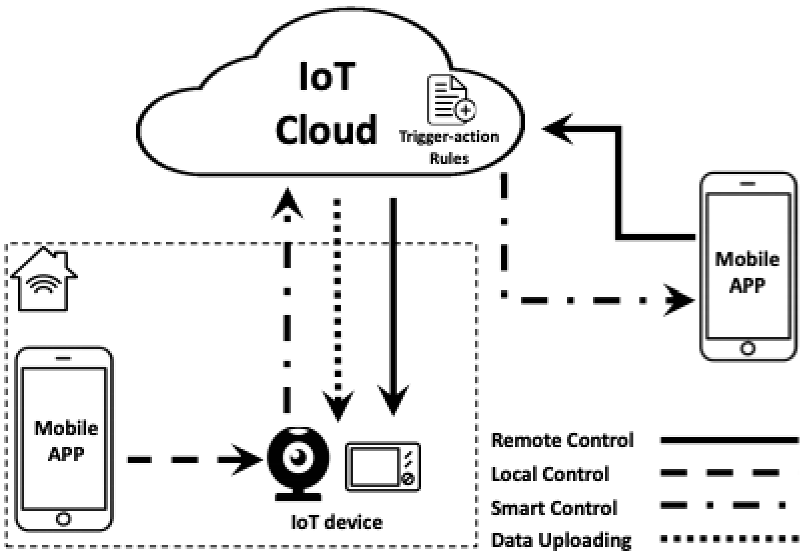


Figure 4. System Model Diagram [46]

3. Related work

This project builds off Giese, Dennis's work [2] on having fun working with IoT devices. The devices used in this project is a modified version of the previous Mi Band variants. The project uses the same background knowledge used in Giese, Dennis's paper. However, this project focuses on performing attacks on the Mi Band 3 [45] rather than the whole IoT eco-system.

The Internet-of-Things BLE Security [3] paper gives a deeper understanding about how the Bluetooth Low Energy [10] characteristics work and how data is transmitted between the BLE device and the device application. Ibrahim, Naseem's [7] paper explains the composition of IoT devices which helps in understanding the hardware used within the device. Once the components are known, Sevier, Seth & Tekeoglu, Ali's paper [10] describes the security analysis within the Bluetooth [39] architecture used in the smart device. This helps in exploiting the easily discoverable aspect of the Mi Band 3.

Verification of Authenticated Firmware [6] is one way to make the Firmware more secure and would be coming from a trusted source and keeps the device safe from attacks. Also, a Lightweight Protocol for IoT authentication [11] explains a much easier way to upload heavy Firmware through a lightweight protocol than the over the air (OTA) [5] update.

4. Threat Classification

In this project, we aim to find vulnerabilities on the Mi Band 3 [5] and classify them into categories. The categories will be based on the root cause for each vulnerability. A little about the Mi Band 3, is that it is a personal wearable device. The main functions include fitness tracking and view alerts from the synchronized smartphone (E.g. Text message, Call alerts, Alarm). The is a very convenient and efficient device, but if they are compromised, one could give away vital information about a person’s fitness details. So, let us look over a few threats the Mi Band 3 [45] faces during its operation.

4.1. Authentication Problems

The Mi Band 3 authenticates with its users through the Mi Fit mobile application. The user must connect the band by choosing the Mi Band 3 option from the Bluetooth settings of the smartphone. Once the connection is established, we must go back to the Mi Fit app and setup the initial configurations. After setting up all the necessary features the band is ready to use. But the other issue with this band is that it authenticates with every device. This means if you have a laptop with BLE stack installed then you can connect with band easily through a few simple Linux commands. There are also many third-party apps which the can band get configured with. Thus, there is no proprietary authentication like other vendors (E.g. Apple Watch configures with Apple Watch app).

4.2. Data Sharing without Consent

Because of the Authentication problems previously explained, the Mi Band 3 shares vital information about the user. The information comprises of Distance Walked, Number of Steps, Calories Burned and Heart rate [29]. So, if someone with a Linux system with BLE stack installed can secretly access the Mi Band 3 and transfer all information to its system. Hence, this issue is like a Man-In-The-Middle attack where a malicious user is between the Mi Band 3 and the Mi Fit app.

5. Experimental Setup

Here we will talk about the various devices we need to look on for the project. As we have discussed in the earlier part of the report about the components [7] that make the IoT [2] architecture. So, in this section we will be discussing more about the components used to emulate an IoT architecture to perform the security analysis. This section will describe the hardware elements that helps to communicate with Mi Band 3 and send commands to it.

5.1. Hardware

In this section, we try to understand the hardware elements used to find the core details (Band MAC address, Band service codes and Send Fake messages). The hardware elements are also used to create a mini IoT Cloud kind of environment. This makes the Mi Band 3 connect with the mini Cloud to accept commands and messages from the mini Cloud created. These hardware elements are easily available as it is present within the systems we use in our daily lives.

5.1.1. x86_64 system with Linux Ubuntu 16.04

This is the most important hardware component used in the project. The best part of this is that Ubuntu [32] 16.04 is open-source and easily available for use. Higher or lower versions of Ubuntu or any Linux operating systems would work without any issues. Linux has many Bluetooth [39] packages that help in packet tracking and analysis stacks like HCItool [36], Bluez stack [53], and Gatttool [38]. Also, present systems are built on x86_64 processors. Hence, there won’t be any compatibility issues. But systems below x86_64 will also work fine.

5.1.2. CSR 4.0 BLE adapter

Now comes the controller medium which is used to send commands and receive data to and from the system and Mi Band 3. When communicating with devices that run on BLE, there is a separate need required to do so. This can be done by altering the in-built Bluetooth hardware present within the laptops or computers. If that does not work, then we can use a high-performance **CSR 4.0 Bluetooth Adapter** [35] with a CSR8510 chip (Figure 5). It is in accord with Bluetooth 4.0 [55] technology standard for longer transmission distance and a more stable transmission rate. With the help of the adaptor, you can add Bluetooth function to your computer without the function. It supports plug and plays technology. And the high-speed data transfer rate can provide you with a smooth and fast experience.

Figure 5. BLE 4.0 CSR Adapter

5.2. Mi Cloud

In today’s world, traditional servers are replaced by Cloud infrastructures. The Cloud architecture being cost effective and having more computation power are a few reasons why Cloud architectures are in high use. Many big companies (Google, Amazon, Microsoft) have their own Cloud infrastructure. Similarly, Xiaomi [15] has their own Cloud service, and this is mostly used for the working of their IoT [2] devices. Xiaomi [1] claims to have the biggest IoT ecosystem worldwide – 85 Million Devices, 800 different models. There can be different vendors, but they are all tied to one ecosystem. It uses the same communication protocol with all devices. Another thing is that different technologies supported. But when it comes to implementation, this aspect can differ from manufacturer to manufacturer and the software quality can be very different.

5.3. Mobile Application

5.3.1. Mi Fit

As in the previous sections we discussed about Mobile app being an integral part in the usage of IoT devices. Thus, the Mi Fit app provided by Xiaomi [1] is the application that authenticates the Mi Band 3 to the user's account. The app is available on both Android and IOS platforms. The Mi Fit application helps in track the user's activity including, analysing sleeping, and evaluating the workouts. Lots of video tutorials keep you motivated and help you build a healthier, more joyful daily routine. This application is also responsible for fetching Firmware update from the Mi Cloud [40] and helps in keeping the band up to date. This app also saves information from the band like the number of steps walked, heart rate, calories burned, and battery life. It also gives alerts about text messages, social media notifications, and call notifications.

5.3.2. nRF Connect

If the hardware method of finding the MAC [30] address of the band does not work, then we can use the nRF Connect application for this purpose. The nRF Connect [34] is an application designed for Bluetooth Low Energy [3] developers. It allows for scanning for BLE devices and communicating with them. The functions which can be executed using this app are scanning for Bluetooth LE devices in range, displaying the RSSI graph [49] with an option to export to CSV or EXCEL, displaying detailed packet history with RSSI, packet changes and advertising intervals, displaying advertisements of multiple devices on a timeline, connecting to multiple devices at the same time, showing device services and characteristics, reading and writing characteristics into the device and many more functionalities.

5.4. Reverse Engineering Tools

Reverse Engineering is the mechanism to debug an object or binary file of a software or application. The debugging is done to look for vulnerabilities in a program so that it cannot be exploited in the future by attackers. Once, we have implemented the authentication of the Mi Band 3 with our Ubuntu system and send requests to import vital data from the band. We would try to understand the reason why the band is synchronizing with a system that does not use the Mi Fit application. For this part of the project, we use four tools namely the APKtool [31], JD-GUI [41] , Eclipse and Ghidra [14] to facilitate our research.

5.4.1. APKTool [45]

Mobile applications on the Android platform comprise of APK (Android Application Package) files. APK files are a compressed form of multiple programs binary files and archived version of java classes. Hence, to convert the APK files into their respective binary files, we need a tool to do this extraction. **APK Tool [31]** can get into the archive, change resources and repack into APK. After the extraction is done, we need to look for a file with an extension of ".fw" or ".bin".

5.4.2. JD- GUI [41]

User interface for editing a Firmware file. JD is a decompiler for the Java programming language. JD is provided as a GUI tool as well as in the form of plug-ins for the Eclipse and IntelliJ IDEA integrated development environments.

5.4.3. Eclipse [42]

API for editing code and open source files. The Eclipse resource model provides API to access and modify files (resources). It also provides mechanisms for efficiently notifying clients of resource changes. The primary method for code to be notified of resources changes is by registering a resource change listener. Listeners are informed what resources are changed and how they changed. The update is proportional to the size of the change and not the size of the workspace.

5.4.4. Ghidra [14]

Ghidra which is an open-source software reverse engineering (SRE) framework developed by NSA's Research Directorate for NSA's cybersecurity mission. It helps analyse malicious code and malware like viruses and can give cybersecurity professionals a better understanding of potential vulnerabilities in their networks and systems. Ghidra has many in-built packages which can convert binary files from various system types and reproduce it to readable code. (E.g. From Hexadecimal numbers to Java class and methods).

6. Methodology

6.1. Mi Band 3 Hardware Identification

In this section we try to understand the hardware components [7] used by the band. The need for doing this is to know more about the processor chip being used. Once, the processor chip is identified, we can try to download the factory settings Firmware provided by the processor vendor. Mi band 3 [5] was released in May 31st, 2018. It has evolved from a basic fitness tracker to a feature rich fitness [19] wearable. Some of the changes are increased screen size, Bluetooth 4.2 BLE, 110 mAh battery capacity and increased waterproof rate of 5 ATM. One of the most notable features introduced is the improved NFC [8] (Near Field Communication) function which helps in quick and easy payments through the band. By looking into the product specification and hardware [5], we found that Mi Band 3 is powered by a DA14580 chip [31]. Unfortunately, the band itself has not debug interface for us to use. To easily **debug the Firmware, we leveraged a development equipped with the same** DA14580 chip.

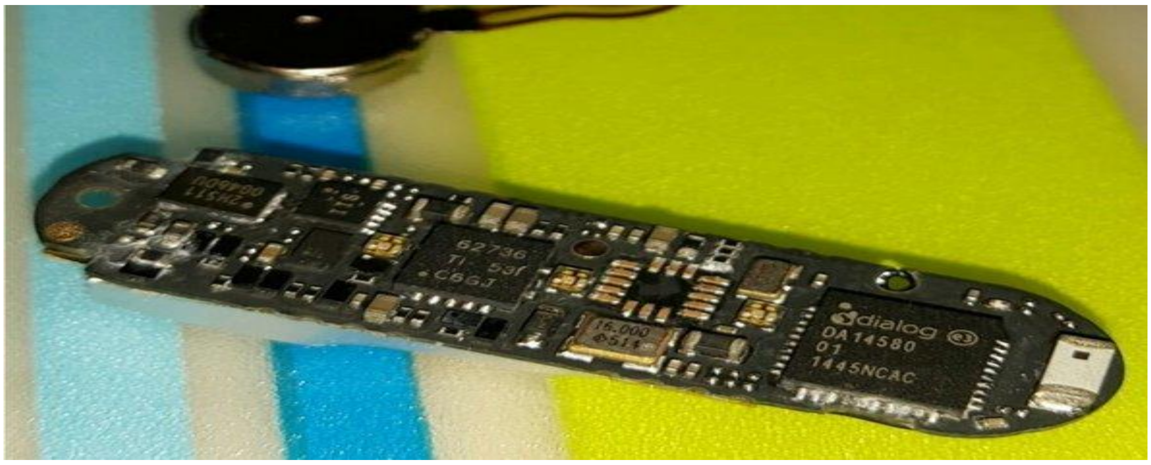


Figure 6. Mi Band 3 Dismantling

6.1.1. Dialog Semiconductor DA14580

Once we have opened the band, we can see that a DA14580 chip [31] is being used as it core processor. This chip is developed by Dialog [26] Semiconductors which specializes in creating hardware for BLE [3] devices. The chip includes a Cortex-M0[43] microcontroller and 42KB RAM._It also integrates radio transceiver and baseband processor for Bluetooth low energy. They can be used as a standalone application processor or as a data [4] pump in hosted systems. The DA14580 supports a flexible memory architecture for storing Bluetooth profiles and custom application code, which can be updated over the air (OTA) [5]. The qualified Bluetooth low energy [10] protocol stack is stored in a dedicated ROM. All software runs on the ARM Cortex-M0[43] processor via a simple scheduler. This makes it very popular in smart wearable market.

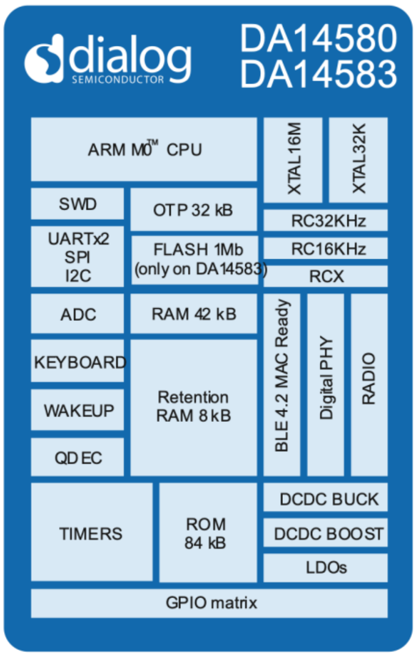


Figure 7. DA14580 circuit diagram

In this section, we try to emulate the whole IoT architecture and perform an attack on the Mi Band 3 [5]. The first thing we would want to exploit is the Authentication [11] aspect of the band. Previously, we have seen that the band connects with almost every device which has Bluetooth within its system. After this, we would try to import data from the band and try to extract as much information as possible like Personal information and daily fitness records. Lastly, we talk about trying to find these vulnerabilities by reverse engineering the Firmware which is installed on the band and try to patch them.

6.2. Authentication System with Band

Once the hardware elements are known, we will try to authenticate the band with our system. (x86_64 Linux Ubuntu 16.04). Through this authentication, we will display an interface which provides options to choose what kind of data is to be imported or viewed. Also provide an option to upload a Custom Firmware which has patches implemented to the factory Firmware.

6.3. Retrieve Data from Band

Now that we have our Mi Band 3 authenticated with our system, we will try to import data or view it on the terminal. The data that we can view is Steps walked, Calories Burned, Reset the time on the band and check the current heartrate of the user. From the heartrate we would come to know whether a user is sedentary or moving.

7. Results

In this section, we illustrate the consequences due to improper authentication implementations in Mi Band 3. In Section 7.1, we show the UI displayed after successful authentication with the device. In Section 7.2, we show the general information like Steps walked, Distance Covered and other vital information [28] present inside the band. In Section 7.3, we can see the live Heartrate [29] of the user wearing the Mi Band 3 through its sensor.

7.1. Vulnerability 1: Mi Band 3 Authentication

7.1.1. System Model

If we were to compare other smart band/watch provides like Apple Inc. We see that any variant of the Apple Watch only connects with an Apple iPhone (smartphone) and Apple Watch application (Mobile application). From this we can make out that the smart watch by Apple does not interact or take connection requests from any other device. But when it comes to the Mi Band 3, the case is slightly complicated. This may be due to the vendors giving less attention to the authentication features of the watch as the price of the band is less expensive. Also, their target audience is almost everyone, so they did not want to complicate the usage of the band for users of any age limit.

7.1.2. Attack Scenario

Firstly, we need to find the MAC address of the Mi Band 3 to start interacting with the it. Thus, we can do this either to using the nRF connect app (Mobile Application) or we can run few Linux commands using the in-built Bluetooth device of your system (Laptop or PC). If we use the nRF Connect app, we can easily get the UUID [44] codes of the band. Using the UUID we can discover the various service methods and easily send commands to the band to authenticate and extract information.

7.1.3. Detailed Attack Process

In this section, we talk about the steps to be performed in order to perform the authentication with the Mi Band 3 using a x86_64 system working on Linux Ubuntu [32] 16.04 operating system. We used an Android app called nRF Connect [34] to sniff the MAC address of the MI 3 band. The same can be done using a BLE 4.0 CSR Adapter [35] and Linux [32] system with necessary Bluetooth packages [55] installed. We download the application from the Android App store. Once we open the application, we click on the option “Scan” to scan all Bluetooth devices around the area. Through this “Scan” we get the MAC address as mentioned in the below Figure 8.

If we cannot download the application, then we can find the MAC address of the smart band using BLE 4.0 CSR Adapter. We connect the adapter to the system via the USB port. We also need a Linux Operating system which contains “HCItool” [36] package which comes preinstalled in Ubuntu with “Bluez” [37] stack and you can consider as the Swiss army knife for Bluetooth in Linux. Once, we have all the requirements, we trace the MAC address using the command “HCItool lescan” in the terminal of the system. This command again provides all the Low-energy Bluetooth devices around the adapter’s range. The below Figure 9 lists all the MAC addresses.

A **UUID [60]** (Universal Unique Identifier) is a 128-bit number used to uniquely identify some object or entity on the Internet. A guaranteed UUID contains a reference to the network address of the host that generated the UUID, a timestamp (a record of the precise time of a transaction), and a randomly generated component. So, a UUID of “0xff01” would be to read DEVICE_INFO [61], UUID of “0xff08” would be to write FIRMWARE_DATA [61] without a response about any updates made to the device.

The following are the steps to fetch UUID for Authentication and pairing with the Band.

- Turn on Bluetooth and HCI log.
- Pair the device to the Xiaomi [1] Android App.
- Turn off Bluetooth.
- Download your **btsnoop_hci.log [2]** to your PC.
- Open it with Wireshark [17].

Find the first ATT protocol [9] request that go to the handle **0x0055** (which represents Anhui Huami Information Technology Co, the company that makes wearable devices and owns the Xiaomi brand). Else, we can simple use the nRF connect [34] Application to view the UUID for authentication like in the image mentioned below. Once we know the UUID for authentication with the Mi Band 3, we can simply feed the values to the authentication code and easily pair with the Mi Band 3. There may be cases where this fails, but this can be resolved using the command "hciconfig hci0 reset" under root privileges are there can be BLE glitches sometimes while trying to authenticate as the Mi Band 3 is continuously discoverable and it may try to interact with other devices around it.

So now we will need to perform the following steps to get some info about how authentication (pairing) works.

- Setting on auth notifications (to get a response) by sending 2 bytes request \x01\x00 to the Destination.

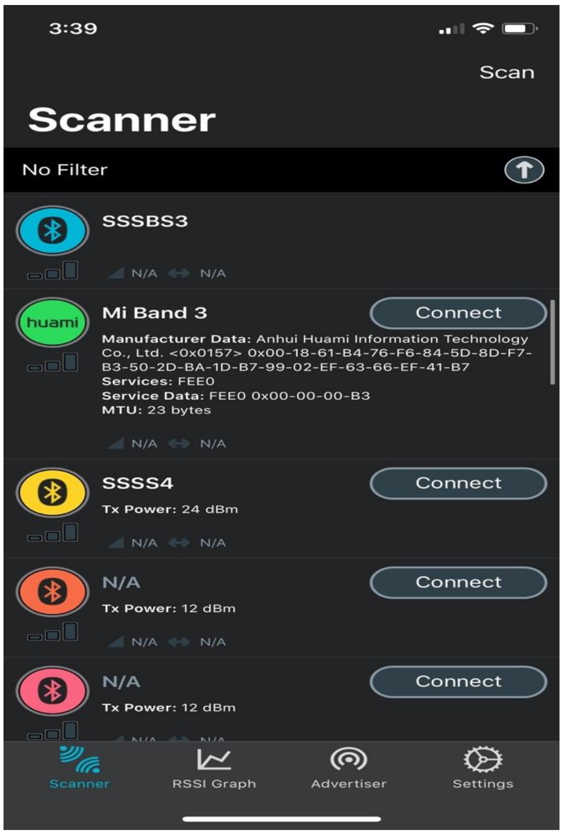
- Send 16 bytes encryption key to the Char with a command and appending to it 2 bytes *\x01\x00 + KEY.*

- Requesting random key from the device with a command by sending 2 bytes *\x02\x00* to the Char.

- Getting random key from the device response (last 16 bytes).

- Encrypting this random number with our 16 bytes key using the AES/ECB/NoPadding **[12]** encryption algorithm (from Crypto.Cipher import AES) and send it back to the Char **(\x03\x00 + encoded data)**

As we can see, no authentication is enforced and anyone with a nRF Connect App or BLE CSR 4.0 adapter [35] and Linux System can authenticate with the Mi Band 3 and compromise it.



3:39

Scan

Scanner

No Filter

SSSBS3

N/A N/A

Mi Band 3 Connect

Manufacturer Data: Anhui Huami Information Technology Co., Ltd. <0x0157> 0x00-18-61-B4-76-F6-84-5D-8D-F7-B3-50-2D-BA-1D-B7-99-02-EF-63-66-EF-41-B7
Services: FEE0
Service Data: FEE0 0x00-00-00-B3
MTU: 23 bytes

N/A N/A

SSSS4 Connect

Tx Power: 24 dBm

N/A N/A

N/A Connect

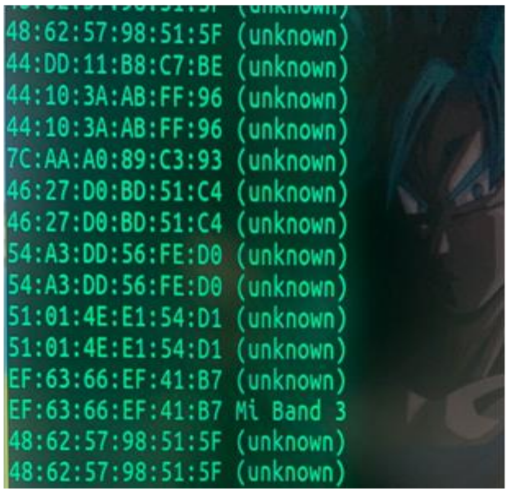
Tx Power: 12 dBm

N/A N/A

N/A Connect

Tx Power: 12 dBm

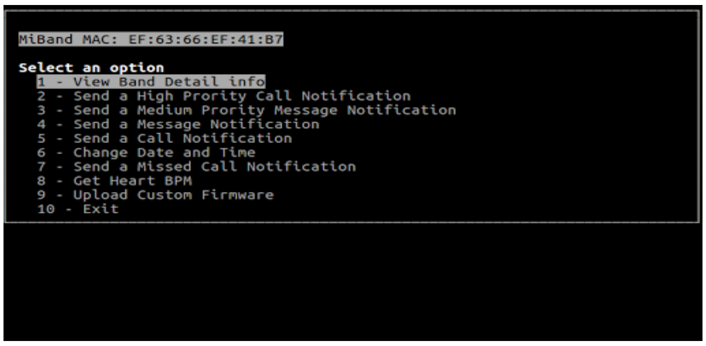
Figure 8. Screenshot of nRF Connect for MAC Address



```
48:62:57:98:51:5F (unknown)
44:DD:11:B8:C7:BE (unknown)
44:10:3A:AB:FF:96 (unknown)
44:10:3A:AB:FF:96 (unknown)
7C:AA:A0:89:C3:93 (unknown)
46:27:D0:BD:51:C4 (unknown)
46:27:D0:BD:51:C4 (unknown)
54:A3:DD:56:FE:D0 (unknown)
54:A3:DD:56:FE:D0 (unknown)
51:01:4E:E1:54:D1 (unknown)
51:01:4E:E1:54:D1 (unknown)
EF:63:66:EF:41:B7 (unknown)
EF:63:66:EF:41:B7 Mi Band 3
48:62:57:98:51:5F (unknown)
48:62:57:98:51:5F (unknown)
```

Figure 9. Terminal screenshot for searching address

Then we run the commands to synchronize and allow the system to play the part of the Mi Fit application. We can see an interactive UI which helps to view various data present in the band.



```
MiBand MAC: EF:63:66:EF:41:B7

Select an option
  1 - View Band Detail info
  2 - Send a High Prority Call Notification
  3 - Send a Medium Prority Message Notification
  4 - Send a Message Notification
  5 - Send a Call Notification
  6 - Change Date and Time
  7 - Send a Missed Call Notification
  8 - Get Heart BPM
  9 - Upload Custom Firmware
  10 - Exit
```

Figure 15. Screenshot of UI on system

7.1.4. Cause Analysis

The main cause for this is the BLE technology. When a device uses BLE, the device interacts with mobile application only for a limited time. For example, when we try to check you hear-rate on the Mi Band 3 and then save the results on the smartphone. This is the only time when the band is active, rest of the time it is in a sleep mode kind of a state. The main advantage of BLE over classic Bluetooth is ultra-low power consumption. But this is also the disadvantage that leads to the band being discoverable most of the times. (E.g. Amazon Echo is always active and is not discoverable). Long story short, BLE is ideal for application which do not require continuous data transfer but highly dependent on battery life.

7.1.5. Defense

From the previous sections we saw how the Mi Band 3 is vulnerable to Authentication from unknown and malicious devices. Here we will talk about methods or techniques which we can implement to prevent this. One such mechanism is to alter the BLE setting within the band to stay non-discoverable once it has been authenticated with the Mi Fit application of the user's smartphone. The other method is to implement a classic Bluetooth chip within the band so that it is always active within the smartphone ecosystem.

7.2. Vulnerability 2: Data Sharing without Consent

7.2.1. System Model

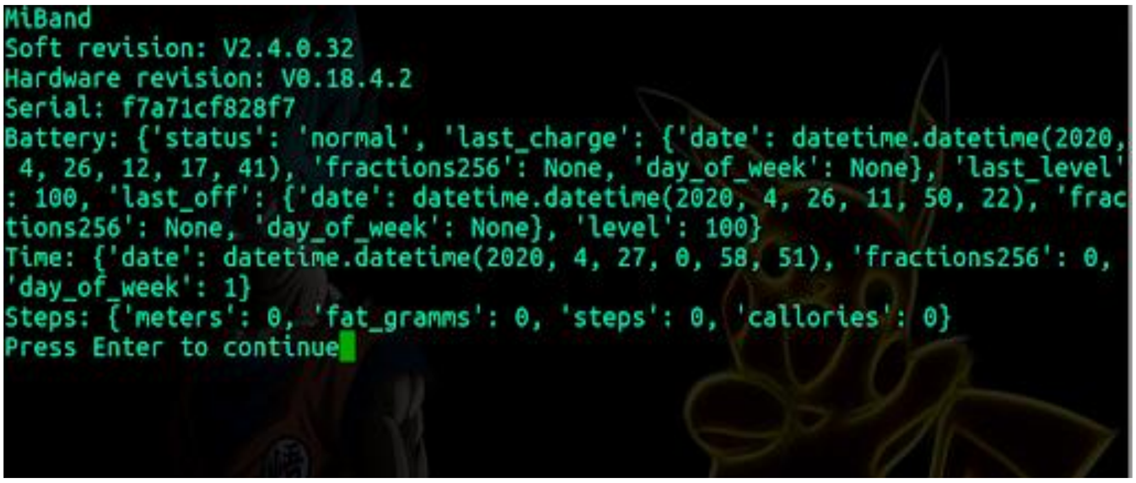
Once we have authenticated with the band through the previous sections. Now it is the time to retrieve data which is already present within the band. All services that interact between the Mi Fit app and the Mi Band 3 is through numeric codes called UUID. A UUID (Universally Unique Identifier) is a 128-bit number used to identify information in computer systems. So, communication between Mi band [5] and Mi Fit [5] App is using many UUIDs so that there won't be much useful information by sniffing BLE data. Hence, we combined sniffing and reversing engineer to analyse the processing of communication. As for characteristics, there are over 20 characteristics and most of them belong to 0xfee0 service. The rest 2 services, 0xfee0 and 0xfee1, are not realized in Mi Fit App as we can see in Figure 10. Through this number we can retrieve information [28] like Hear-rate, Number of steps walked, GPS location and many other vital information. So, we need to send the UUID through a sequence of commands from the Linux System which is used to authenticate with the band. Then we will retrieve the raw data from the band and store it within our system.

7.2.2. Attack Scenario

As some in Figure 10, the band provides General Attribute profile (GATT) [38], that defines how communication/data transfer between client and server. This short piece of information that is being sent and received are called attributes. BLE [3] has few key concepts, such as profiles, services & characteristics. Services are the set of provided features and associated behaviors to interact with the peripheral. Each service contains a collection of characteristics. Characteristics are defined attribute types that contain a single logical value. In the Figure 10 we can see that with the help of Gatttool we can find primary and other (Figure 11) UUID's required to contact with band to fetch the raw data. So how do we know which number corresponds to which function. We can do this by referencing each value from the terminal (Figure 10 and 11) and match in the corresponding number in Figure 12. Once we know which UUID corresponds to which function, we can fetch the respective data needed and display it on our terminal screen (Figure 15).

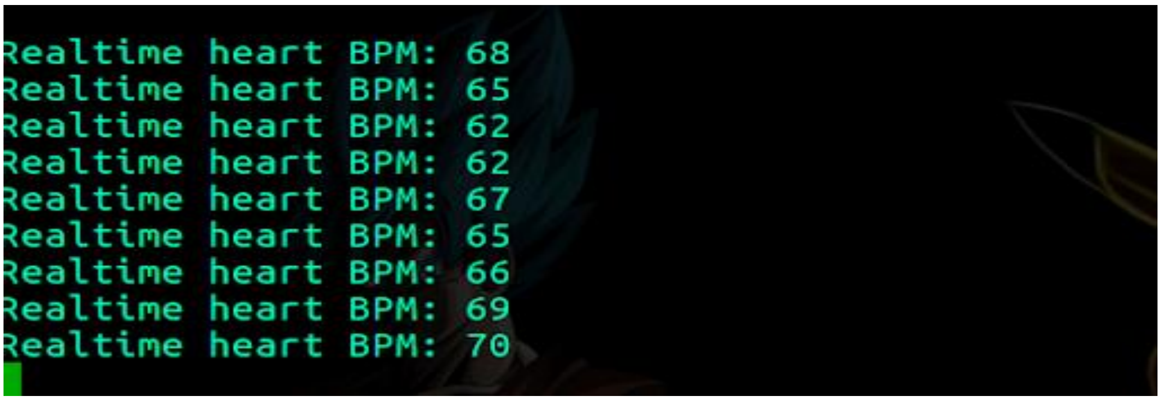
7.2.3. Detailed Attack Process

When we choose the first option of View Band Detail Info, we can see that it displays the band name, Software version, Serial Number of the Device, Battery Status, Date and Time. This option pretty much covers the basic information and daily activity. When we choose the option Get Heart BPM, we get the live Heartrate of the user wearing the band and will run till we do not stop the execution. Through this we can anticipate whether the user is sedentary or moving around through the rise or drop of the heartrate.



```
MiBand
Soft revision: V2.4.0.32
Hardware revision: V0.18.4.2
Serial: f7a71cf828f7
Battery: {'status': 'normal', 'last_charge': {'date': datetime.datetime(2020,
 4, 26, 12, 17, 41), 'fractions256': None, 'day_of_week': None}, 'last_level'
: 100, 'last_off': {'date': datetime.datetime(2020, 4, 26, 11, 50, 22), 'frac
tions256': None, 'day_of_week': None}, 'level': 100}
Time: {'date': datetime.datetime(2020, 4, 27, 0, 58, 51), 'fractions256': 0,
'day_of_week': 1}
Steps: {'meters': 0, 'fat_gramms': 0, 'steps': 0, 'callories': 0}
Press Enter to continue
```

Figure 16. Information from Band



```
Realtime heart BPM: 68
Realtime heart BPM: 65
Realtime heart BPM: 62
Realtime heart BPM: 62
Realtime heart BPM: 67
Realtime heart BPM: 65
Realtime heart BPM: 66
Realtime heart BPM: 69
Realtime heart BPM: 70
```

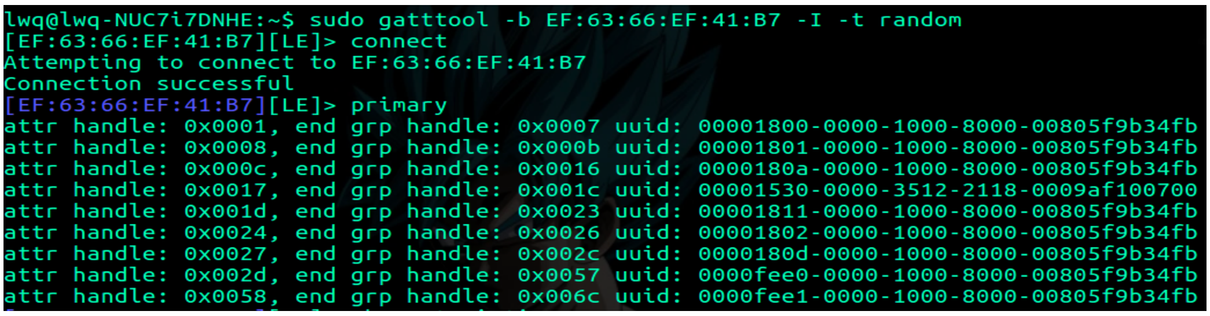
Figure 17. Information of Users Heartbeat

7.2.4. Cause Analysis

As we can see that Linux provides us with built-in Bluetooth stacks and packages like HCItool, GATT and Bluez [37]. All these packages were built to work on classic Bluetooth and BLE technology using devices. Therefore, there no need to study each Bluetooth technology type to write a specific piece of code for it. (E.g. To find MAC address has built-in function called "lescan" under HCItool package). Hence, we can say that if a device has Bluetooth, it can be easily scanned with the help of these packages provided by Linux.

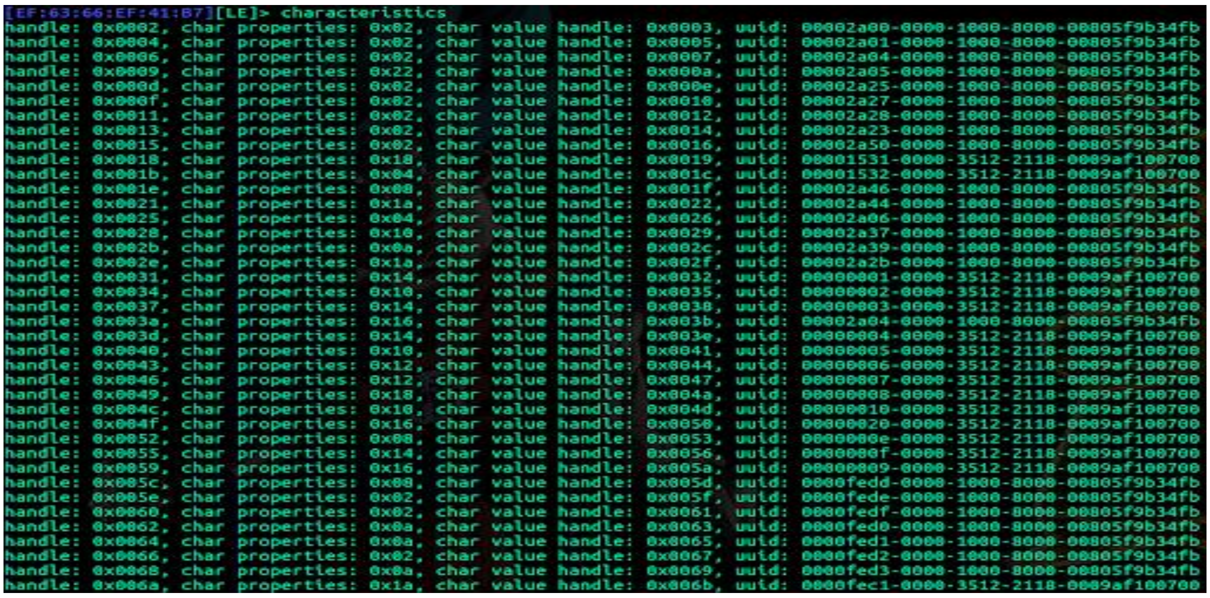
7.2.5. Defense

It is kind of hard to impose a defence mechanism for this issue as Linux is open-source and easily available to any user's disposal. The one thing that IoT [2] vendors could do is to add restrictions on their technologies in terms of hardware alterations within the Bluetooth chip and software restrictions through code. This will make the device not visible to any Bluetooth scans done by systems using HCI tool packages.



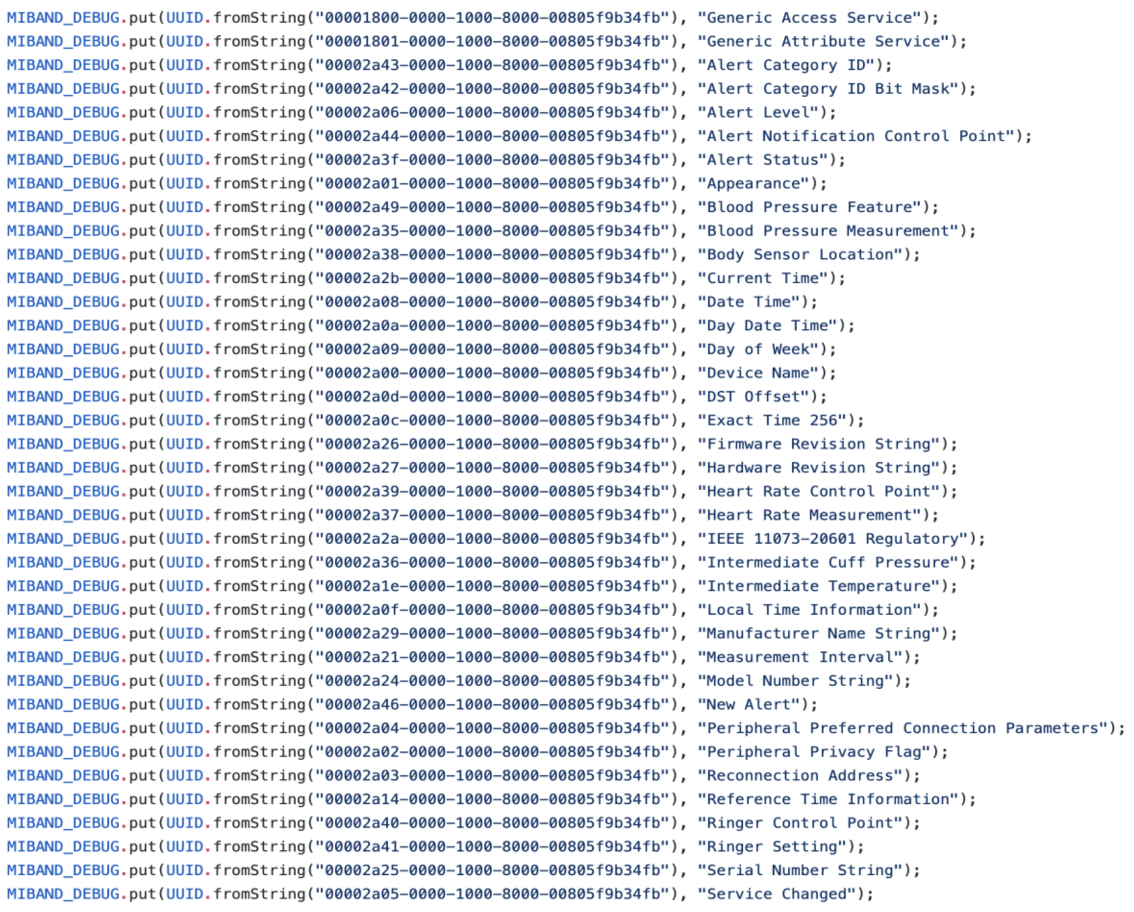
```
lwq@lwq-NUC7i7DNHE:~$ sudo gatttool -b EF:63:66:EF:41:B7 -I -t random
[EF:63:66:EF:41:B7][LE]> connect
Attempting to connect to EF:63:66:EF:41:B7
Connection successful
[EF:63:66:EF:41:B7][LE]> primary
attr handle: 0x0001, end grp handle: 0x0007 uuid: 00001800-0000-1000-8000-00805f9b34fb
attr handle: 0x0008, end grp handle: 0x000b uuid: 00001801-0000-1000-8000-00805f9b34fb
attr handle: 0x000c, end grp handle: 0x0016 uuid: 0000180a-0000-1000-8000-00805f9b34fb
attr handle: 0x0017, end grp handle: 0x001c uuid: 00001530-0000-3512-2118-0009af100700
attr handle: 0x001d, end grp handle: 0x0023 uuid: 00001811-0000-1000-8000-00805f9b34fb
attr handle: 0x0024, end grp handle: 0x0026 uuid: 00001802-0000-1000-8000-00805f9b34fb
attr handle: 0x0027, end grp handle: 0x002c uuid: 0000180d-0000-1000-8000-00805f9b34fb
attr handle: 0x002d, end grp handle: 0x0057 uuid: 0000fee0-0000-1000-8000-00805f9b34fb
attr handle: 0x0058, end grp handle: 0x006c uuid: 0000fee1-0000-1000-8000-00805f9b34fb
```

Figure 10. Terminal screenshot for finding primary function ID's (UUID)



```
[EF:63:66:EF:41:B7][LE]> characteristics
handle: 0x0002, char properties: 0x02, char value handle: 0x0003, uuid: 00002a00-0000-1000-8000-00805f9b34fb
handle: 0x0004, char properties: 0x02, char value handle: 0x0005, uuid: 00002a01-0000-1000-8000-00805f9b34fb
handle: 0x0006, char properties: 0x02, char value handle: 0x0007, uuid: 00002a04-0000-1000-8000-00805f9b34fb
handle: 0x0009, char properties: 0x22, char value handle: 0x000a, uuid: 00002a05-0000-1000-8000-00805f9b34fb
handle: 0x000d, char properties: 0x02, char value handle: 0x000e, uuid: 00002a25-0000-1000-8000-00805f9b34fb
handle: 0x000f, char properties: 0x02, char value handle: 0x0010, uuid: 00002a27-0000-1000-8000-00805f9b34fb
handle: 0x0011, char properties: 0x02, char value handle: 0x0012, uuid: 00002a28-0000-1000-8000-00805f9b34fb
handle: 0x0013, char properties: 0x02, char value handle: 0x0014, uuid: 00002a23-0000-1000-8000-00805f9b34fb
handle: 0x0015, char properties: 0x02, char value handle: 0x0016, uuid: 00002a50-0000-1000-8000-00805f9b34fb
handle: 0x0018, char properties: 0x18, char value handle: 0x0019, uuid: 00001531-0000-3512-2118-0009af100700
handle: 0x001b, char properties: 0x04, char value handle: 0x001c, uuid: 00001532-0000-3512-2118-0009af100700
handle: 0x001e, char properties: 0x08, char value handle: 0x001f, uuid: 00002a46-0000-1000-8000-00805f9b34fb
handle: 0x0021, char properties: 0x1a, char value handle: 0x0022, uuid: 00002a44-0000-1000-8000-00805f9b34fb
handle: 0x0025, char properties: 0x04, char value handle: 0x0026, uuid: 00002a06-0000-1000-8000-00805f9b34fb
handle: 0x0028, char properties: 0x10, char value handle: 0x0029, uuid: 00002a37-0000-1000-8000-00805f9b34fb
handle: 0x002b, char properties: 0x0a, char value handle: 0x002c, uuid: 00002a39-0000-1000-8000-00805f9b34fb
handle: 0x002e, char properties: 0x1a, char value handle: 0x002f, uuid: 00002a2b-0000-1000-8000-00805f9b34fb
handle: 0x0031, char properties: 0x14, char value handle: 0x0032, uuid: 00000001-0000-3512-2118-0009af100700
handle: 0x0034, char properties: 0x10, char value handle: 0x0035, uuid: 00000002-0000-3512-2118-0009af100700
handle: 0x0037, char properties: 0x14, char value handle: 0x0038, uuid: 00000003-0000-3512-2118-0009af100700
handle: 0x003a, char properties: 0x16, char value handle: 0x003b, uuid: 00002a04-0000-1000-8000-00805f9b34fb
handle: 0x003d, char properties: 0x14, char value handle: 0x003e, uuid: 00000004-0000-3512-2118-0009af100700
handle: 0x0040, char properties: 0x10, char value handle: 0x0041, uuid: 00000005-0000-3512-2118-0009af100700
handle: 0x0043, char properties: 0x12, char value handle: 0x0044, uuid: 00000006-0000-3512-2118-0009af100700
handle: 0x0046, char properties: 0x12, char value handle: 0x0047, uuid: 00000007-0000-3512-2118-0009af100700
handle: 0x0049, char properties: 0x18, char value handle: 0x004a, uuid: 00000008-0000-3512-2118-0009af100700
handle: 0x004c, char properties: 0x10, char value handle: 0x004d, uuid: 00000010-0000-3512-2118-0009af100700
handle: 0x004f, char properties: 0x16, char value handle: 0x0050, uuid: 00000020-0000-3512-2118-0009af100700
handle: 0x0052, char properties: 0x08, char value handle: 0x0053, uuid: 0000000e-0000-3512-2118-0009af100700
handle: 0x0055, char properties: 0x14, char value handle: 0x0056, uuid: 0000000f-0000-3512-2118-0009af100700
handle: 0x0059, char properties: 0x16, char value handle: 0x005a, uuid: 00000009-0000-3512-2118-0009af100700
handle: 0x005c, char properties: 0x08, char value handle: 0x005d, uuid: 0000fedd-0000-1000-8000-00805f9b34fb
handle: 0x005e, char properties: 0x02, char value handle: 0x005f, uuid: 0000fede-0000-1000-8000-00805f9b34fb
handle: 0x0060, char properties: 0x02, char value handle: 0x0061, uuid: 0000fedf-0000-1000-8000-00805f9b34fb
handle: 0x0062, char properties: 0x0a, char value handle: 0x0063, uuid: 0000fed0-0000-1000-8000-00805f9b34fb
handle: 0x0064, char properties: 0x0a, char value handle: 0x0065, uuid: 0000fed1-0000-1000-8000-00805f9b34fb
handle: 0x0066, char properties: 0x02, char value handle: 0x0067, uuid: 0000fed2-0000-1000-8000-00805f9b34fb
handle: 0x0068, char properties: 0x0a, char value handle: 0x0069, uuid: 0000fed3-0000-1000-8000-00805f9b34fb
handle: 0x006a, char properties: 0x1a, char value handle: 0x006b, uuid: 0000fec1-0000-3512-2118-0009af100700
```

Figure 11. Terminal screenshot for finding other function ID's (UUID)



```
MIBAND_DEBUG.put(UUID.fromString("00001800-0000-1000-8000-00805f9b34fb"), "Generic Access Service");
MIBAND_DEBUG.put(UUID.fromString("00001801-0000-1000-8000-00805f9b34fb"), "Generic Attribute Service");
MIBAND_DEBUG.put(UUID.fromString("00002a43-0000-1000-8000-00805f9b34fb"), "Alert Category ID");
MIBAND_DEBUG.put(UUID.fromString("00002a42-0000-1000-8000-00805f9b34fb"), "Alert Category ID Bit Mask");
MIBAND_DEBUG.put(UUID.fromString("00002a06-0000-1000-8000-00805f9b34fb"), "Alert Level");
MIBAND_DEBUG.put(UUID.fromString("00002a44-0000-1000-8000-00805f9b34fb"), "Alert Notification Control Point");
MIBAND_DEBUG.put(UUID.fromString("00002a3f-0000-1000-8000-00805f9b34fb"), "Alert Status");
MIBAND_DEBUG.put(UUID.fromString("00002a01-0000-1000-8000-00805f9b34fb"), "Appearance");
MIBAND_DEBUG.put(UUID.fromString("00002a49-0000-1000-8000-00805f9b34fb"), "Blood Pressure Feature");
MIBAND_DEBUG.put(UUID.fromString("00002a35-0000-1000-8000-00805f9b34fb"), "Blood Pressure Measurement");
MIBAND_DEBUG.put(UUID.fromString("00002a38-0000-1000-8000-00805f9b34fb"), "Body Sensor Location");
MIBAND_DEBUG.put(UUID.fromString("00002a2b-0000-1000-8000-00805f9b34fb"), "Current Time");
MIBAND_DEBUG.put(UUID.fromString("00002a08-0000-1000-8000-00805f9b34fb"), "Date Time");
MIBAND_DEBUG.put(UUID.fromString("00002a0a-0000-1000-8000-00805f9b34fb"), "Day Date Time");
MIBAND_DEBUG.put(UUID.fromString("00002a09-0000-1000-8000-00805f9b34fb"), "Day of Week");
MIBAND_DEBUG.put(UUID.fromString("00002a00-0000-1000-8000-00805f9b34fb"), "Device Name");
MIBAND_DEBUG.put(UUID.fromString("00002a0d-0000-1000-8000-00805f9b34fb"), "DST Offset");
MIBAND_DEBUG.put(UUID.fromString("00002a0c-0000-1000-8000-00805f9b34fb"), "Exact Time 256");
MIBAND_DEBUG.put(UUID.fromString("00002a26-0000-1000-8000-00805f9b34fb"), "Firmware Revision String");
MIBAND_DEBUG.put(UUID.fromString("00002a27-0000-1000-8000-00805f9b34fb"), "Hardware Revision String");
MIBAND_DEBUG.put(UUID.fromString("00002a39-0000-1000-8000-00805f9b34fb"), "Heart Rate Control Point");
MIBAND_DEBUG.put(UUID.fromString("00002a37-0000-1000-8000-00805f9b34fb"), "Heart Rate Measurement");
MIBAND_DEBUG.put(UUID.fromString("00002a2a-0000-1000-8000-00805f9b34fb"), "IEEE 11073-20601 Regulatory");
MIBAND_DEBUG.put(UUID.fromString("00002a36-0000-1000-8000-00805f9b34fb"), "Intermediate Cuff Pressure");
MIBAND_DEBUG.put(UUID.fromString("00002a1e-0000-1000-8000-00805f9b34fb"), "Intermediate Temperature");
MIBAND_DEBUG.put(UUID.fromString("00002a0f-0000-1000-8000-00805f9b34fb"), "Local Time Information");
MIBAND_DEBUG.put(UUID.fromString("00002a29-0000-1000-8000-00805f9b34fb"), "Manufacturer Name String");
MIBAND_DEBUG.put(UUID.fromString("00002a21-0000-1000-8000-00805f9b34fb"), "Measurement Interval");
MIBAND_DEBUG.put(UUID.fromString("00002a24-0000-1000-8000-00805f9b34fb"), "Model Number String");
MIBAND_DEBUG.put(UUID.fromString("00002a46-0000-1000-8000-00805f9b34fb"), "New Alert");
MIBAND_DEBUG.put(UUID.fromString("00002a04-0000-1000-8000-00805f9b34fb"), "Peripheral Preferred Connection Parameters");
MIBAND_DEBUG.put(UUID.fromString("00002a02-0000-1000-8000-00805f9b34fb"), "Peripheral Privacy Flag");
MIBAND_DEBUG.put(UUID.fromString("00002a03-0000-1000-8000-00805f9b34fb"), "Reconnection Address");
MIBAND_DEBUG.put(UUID.fromString("00002a14-0000-1000-8000-00805f9b34fb"), "Reference Time Information");
MIBAND_DEBUG.put(UUID.fromString("00002a40-0000-1000-8000-00805f9b34fb"), "Ringer Control Point");
MIBAND_DEBUG.put(UUID.fromString("00002a41-0000-1000-8000-00805f9b34fb"), "Ringer Setting");
MIBAND_DEBUG.put(UUID.fromString("00002a25-0000-1000-8000-00805f9b34fb"), "Serial Number String");
MIBAND_DEBUG.put(UUID.fromString("00002a05-0000-1000-8000-00805f9b34fb"), "Service Changed");
```

Figure 12. UUID Interpretation list

8. Proposed Defense

Based on our research result, in this section, we propose some countermeasures for safeguarding the security [22] of Mi band 3. We hope to reverse engineer the Firmware of the MI Band 3 and make binary modifications [36] to patch the security holes. Our first step is extracting the Firmware image. Then we analyze it and apply patches. For example, we can inject code to perform additional authentications. We can also disable unauthorized Firmware update so that a malicious user cannot inject code. To prevent the attacks of Unauthorized Authentication and Data Sharing without Consent could be resolved by patching the bugs in the Firmware [16] which causes this. Firmware usually are made of multiple binary files and can be easily recognized through its extension (E.g. .fw or .bin). Through references we understand that the Firmware [6] used by the Mi Band 3 is named as Mili_wuhan.fw and Mili_wuhan.res. The ".fw" file consists of the functions and methods that help in the functioning of the band. The ".res" file consists of all logos and images used to make the band more visually pleasant.

8.1. Steps to Retrieve Firmware

We can silently overwrite existing Firmware by choosing the option Upload Custom Firmware [17] and cause even more damage to the user's device through information theft. This is mainly due to the hardware manufacturers not cryptographically signing [17] the Firmware embedded into the device nor include authentication features in the devices can recognize if the Firmware being pushed from an authentic vendor or application.

There were times where if the Mi Band 3 was corrupted or bricked due to repeated authentications. If the band is connected to the Mi Fit application, it uploads the Firmware automatically. Through this intuition, we proceeded to studying the Mi Fit application to see how the it would upload the Firmware on to the band. Apkpure is the best web resource to download any Android application and its corresponding APK files. After downloading the APK file, we needed to extract the files within the APK file. For this, we can use the APK tool to extract files from the application. There are many tools available online for extraction and they are free to use. After extraction, there were many files many files and resources present within the application. But interestingly, in the "**assets"** directory of the APK file, we can see many files with ".res" and ".fw" extensions. After a few Google searches, we can understand that these extensions mean that they are the resource and Firmware files for a device. The ".fw" file has the Firmware, string resource, software version number and much more. The ".res" file has all the resource images and animations which we can see on the band during alerts.

8.2. Analyze Firmware using Ghidra

Here we try to understand about the Firmware retrieved from the APKtool using Ghidra. This is an open-source tool provided by NSA which can be used for Firmware analysis and patching it with the required fix. But the problem when we try to investigate the firmware is its high Obfuscation within the binary files. The Xiaomi Mi Fit app [5] is obfuscated, making it difficult for attackers to reverse-engineer its functionality. Furthermore, 2-factor authentication [11] via the "Xiaomi Authenticator" app is integrated. Apart from Apple, Xiaomi is the only vendor in our test that has implemented this security feature. In the app-data folder, a protected area to which only the respective app has access, a detailed log including a relatively easy to read the description of the API is written. However, this is only dangerous on rooted devices, which is why this is not considered negatively in the evaluation.

8.3. Analyze Firmware using Binwalk

Binwalk [27] is one of the ways to analyse binary files. Binwalk stack is available on every Linux platform and has many options for binary analysis. One of the most important options is -E extension of binwalk [27] which is finding the Entropy for the file. Entropy is showing how secured and encrypted a file is making it difficult to reverse engineer. The graphs especially show that the Firmware provided by the vendor company is highly encrypted. We also used Ghidra [14] to reverse engineer the image. While viewing the Firmware file, the start point of the program and other important details were not easily detectable as the code is very obfuscated by the Firmware developers as to prevent any sort of reverse engineering.

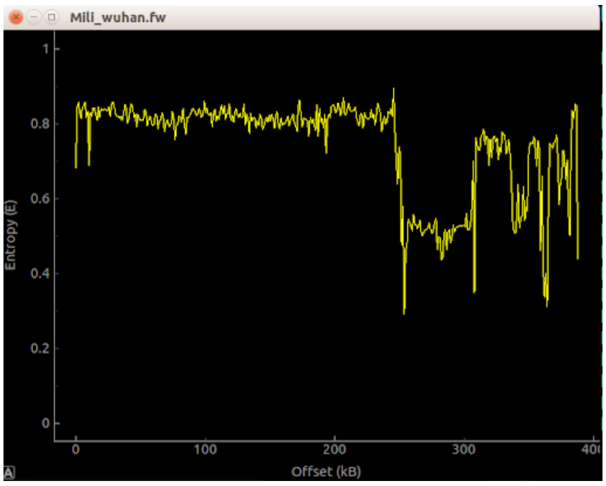


Figure 13. Entropy Graph for latest Firmware

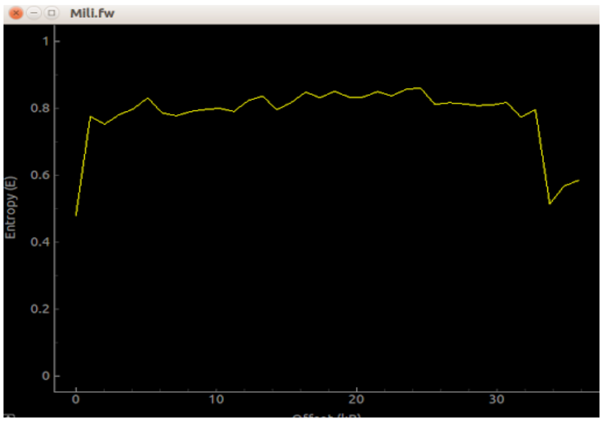


Figure 14. Entropy Graph for old Firmware

8.4. Obstacles Faced during Analysis

The Mi Band 3 receives it firmware update through the Mi Fit application. This allows us to easily obtain the firmware. However, the encryption of the firmware imposes a challenge for me to reverse engineer the firmware to find out the vulnerabilities. The encryption level within a Firmware [24] is made high so that others cannot manipulate it and create new copies (E.g. Like working on Trademarked Software). On the one hand, this protects the IP of manufactures. On the other hand, it also disables white-hat hackers to manipulate the firmware for security enhancement. If it was easy to reverse engineer the Firmware, then we could have found out the reason for the authentication vulnerabilities within the Firmware and apply the corresponding patches to mitigate the problem [20].

9. Accomplishments and Issues

Starting from the Spring semester, most of the design specifications were implemented successfully. The band leaks essential information to the unauthorized system with a BLE connection making it vulnerable. The language used is Python [33] which has many packages ready for import for using BLE technologies [3] and making it less tedious due to code simplicity. The issues come when trying to fix these vulnerabilities as the Firmware from the vendors is heavily obfuscated making it hard to find entry points to the program and other essential parts of the binary.

10. Conclusion

Mi Band is a very good device when it comes to durability, functionalities, and cost-efficiency. Even the Firmware update over the air [5] is a futuristic feature for a small band like it and removes the task of connecting the band to the mobile using wires for just updating the Firmware. I tried to display how the band is easily discoverable through BLE adapters and anyone with the right knowledge can take control of the device. This can also lead to uploading malicious Firmware that can steal sensitive information from the band. This is mainly due to the hardware manufacturers not cryptographically signing the Firmware embedded into the device nor include authentication features in the devices can recognize if the Firmware being pushed from an authentic vendor or application. The device willingly accepts the Firmware from any device. The solution for this could be that hardware manufacturers should design Firmware and Firmware update they distribute to be cryptographically signed.

11. References

[1] Xiaomi and the Future: https://medium.com/swlh/xiaomi-and-the-future-of-iot-2efecda5ef4b

[2] Giese, Dennis. (2018). Having fun with IoT: Reverse Engineering and Hacking of Xiaomi IoT Devices.

[3] Thomas Chiu, David Calero Luis, and Vinesh Jethva. 2017. Internet of Things BLE Security. In Proceedings of the 6th Annual Conference on Research in Information Technology (RIIT '17). Association for Computing Machinery, New York, NY, USA, 37. DOI:https://doi.org/10.1145/3125649.3125656

[4] Gołosz, Mateusz & Mrozek, Dariusz. (2019). Exploration of Data from Smart Bands in the Cloud and on the Edge – The Impact on the Data Storage Space. 10.1007/978-3-030-22744-9_47.

[5] Mi Band 3: https://www.mi.com/global/mi-band-3/specs

[6] Muduli, Sujit & Subramanyan, Pramod & Ray, Sayak. (2019). Verification of Authenticated Firmware Loaders. 110-119. 10.23919/FMCAD.2019.8894262.

[7] Ibrahim, Naseem. (2019). Composition of Things in the Internet of Things. 1168-1173. 10.1109/CSCI49370.2019.00220.

[8] Majumder, Alak & Ghosh, Shirsha & Goswami, Joyeeta & Bhattacharyya, Bidyut. (2017). NFC in IoT-Based Payment Architecture. 10.1201/9781315156026-12.

[9] ATT Protocol: https://epxx.co/artigos/bluetooth_gatt.html

[10] Sevier, Seth & Tekeoglu, Ali. (2019). Analyzing the Security of Bluetooth Low Energy. 1-5. 10.23919/ELINFOCOM.2019.8706457.

[11] Tarish, Hiba. (2018). Proposed Lightweight Protocol for IoT Authentication. 10.13140/RG.2.2.28741.70881.

[12] Almolhis, Nawaf & Alashjaee, Abdullah & Duraibi, Salahaldeen & Alqahtani, Fahad & Nour, Ahmed. (2020). The Security Issues in IoT - Cloud: A Review. 191-196. 10.1109/CSPA48992.2020.9068693.

[13] Pflanzner, Tamas & Kertész, Attila. (2016). A survey of IoT Cloud providers. 730-735. 10.1109/MIPRO.2016.7522237. //

[14] Rohleder, Roman. (2019). Hands-On Ghidra - A Tutorial about the Software Reverse Engineering Framework. 77-78. 10.1145/3338503.3357725.

[15] Song, Yao & Luximon, Yan & Leong, Benny & Qin, Zhenzhen. (2019). The E-Commerce Performance of Internet of Things (IoT) in Disruptive Innovation: Case of XiaoMi. 188-192. 10.1145/3374549.3374557.

[16] Smith, Joshua. (2019). Case Analysis of Firmware Vulnerabilities and Exploitation.

[17] Wireshark: https://www.wireshark.org/

[18] Bohli, Jens-Matthias & Dietrich, Aljoscha & Petrlic, Ronald & Sorge, Christoph. (2017). A Comparison of Payment Schemes for the IoT.

[19] Pal, Debajyoti & Tassanaviboon, Anuchart & Arpnikanondt, Chonlameth & Papasratorn, Borworn. (2019). Quality of Experience of Smart-Wearables: From Fitness-Bands to Smartwatches. IEEE Consumer Electronics Magazine. 9. 49-53. 10.1109/MCE.2019.2941462.

[20] Xie, Wei & Jiang, Yikun & Tang, Yong & Ding, Ning & Gao, Yuanming. (2017). Vulnerability Detection in IoT Firmware: A Survey. 769-772. 10.1109/ICPADS.2017.00104.

[21] Bauer, Martin & Bui, Nicola & Jardak, Christine & Nettsträter, Andreas. (2013). The IoT ARM reference manual. 10.1007/978-3-642-40403-0_9.

[22] In the security check: https://www.iot-tests.org/2018/08/in-the-security-check-xiaomi-mi-band-2/

[23] Li, Wenhao & Chen, Haibo. (2019). Research on ARM TrustZone. GetMobile: Mobile Computing and Communications. 22. 17-22. 10.1145/3308755.3308761.

[24] Costin, Andrei & Zaddach, Jonas & Francillon, Aurelien & Balzarotti, Davide. (2014). A Large-Scale Analysis of the Security of Embedded Firmwares.

[25] What is the Internet of Things: https://www.wired.co.uk/article/Internet-of-things-what-is-explained-iot
[26] Dialog SmartBond DA14580: https://www.dialog-semiconductor.com/products/connectivity/bluetooth-low-energy/smartbond-da14580-and-da14583

[27] Binwalk Tools: https://tools.kali.org/forensics/binwalk

[28] Celik, Z. Berkay & Babun, Leonardo & Sikder, Amit Kumar & Aksu, Hidayet & Tan, Gang & McDaniel, Patrick & Uluagac, Selcuk. (2018). Sensitive Information Tracking in Commodity IoT.

[29] Eberz, Simon & Patané, Andrea & Paoletti, Nicola & Kwiatkowska, Marta & Roeschlin, Marc & Martinovic, Ivan. (2017). Broken Hearted: How to Attack ECG Biometrics. 10.14722/ndss.2017.23408.

[30] Asija, Monika. (2016). MAC Address. IRA-International Journal of Technology & Engineering (ISSN 2455-4480). 3. 10.21013/jte.v3.n1.p5.

[31] Online APK tool: http://www.javadecompilers.com/apktool

[32] Ubuntu Linux 16.04: https://releases.ubuntu.com/16.04/

[33] Python: https://www.python.org/

[34] nRF Connect: https://www.nordicsemi.com/Software-and-tools/Development-Tools/nRF-Connect-for-desktop

[35] BLE CSR 4.0 Adapter: https://retrosystemsrevival.blogspot.com/2019/08/csr8510-a10-bluetooth-adapter-driver.html

[36] HCITOOL: https://linux.die.net/man/1/HCItool

[37] Bluez Stack: http://www.Bluez.org/

[38] Gatttool: https://helpmanual.io/help/gatttool/

[39] Bluetooth Characteristics: https://www.bluetooth.com/specifications/gatt/characteristics/

[40] Mi Cloud: https://i.mi.com/?_locale=en

[41] JD-GUI: http://java-decompiler.github.io/

[42] Eclipse: https://www.eclipse.org/eclipseide/

[43] Cortex-M4: https://developer.arm.com/ip-products/processors/cortex-m/cortex-m4

[44] UUID: https://tools.ietf.org/html/rfc4122

[45] Mi Band 3 Protocol Analyze: http://changy-.github.io/articles/xiao-mi-band-protocol-analyze.html

[46] Zhou, Wei & Cao, Chen & Huo, Dongdong & Cheng, Kai & Zhang, Lan & Guan, Le & Liu, Tao & Zheng, Yaowen & Zhang, Yuqing & Sun, Limin & Wang, Yazhe & Liu, Peng. (2019). Logic Bugs in IoT Platforms and Systems: A Review.