proceedings

Proceedings

Assessment of Fitness Tracker Security: A Case of Study [†]

Florina Almenares Mendoza [1,*], Lucía Alonso [2], Andrés Marín López [1], Daniel Díaz Sánchez [1] and Patricia Arias Cabarcos [1,‡]

1 University Carlos III de Madrid, Avda, de la Universidad 30, 28911 Leganés, Spain; amarin@it.uc3m.es (A.M.L.); dds@it.uc3m.es (D.D.S.), ariasp@it.uc3m.es (P.A.C.)
2 Alumni University Carlos III of Madrid, 28911 Madrid, Spain; 100355904@alumnos.uc3m.es
* Correspondence: florina@it.uc3m.es
† Presented at the 12th International Conference on Ubiquitous Computing and Ambient Intelligence (UCAmI 2018), Punta Cana, Dominican Republic, 4–7 December 2018.
‡ Currently supported by an Alexander von Humboldt fellowship at Universität Mannheim.

Published: 26 October 2018

Abstract: The wearable industry has experienced a notable growth over the last decade, especially in fitness or e-health trackers. These trackers bring new functionalities that require collecting a great amount of sensitive information about the user. This fact has made fitness trackers the target of deliberate attacks, e.g., eavesdropping, unauthorized account access, fake firmware update, and so on. For this reason, this paper describes a vulnerability study on one of the most popular fitness trackers in 2017, together with the mobile application associated to the tracker. The study results show what vulnerabilities of the communications among agents (i.e., wearable device, mobile application and server) could put at risk users sensitive information and privacy.

Keywords: fitness tracker; wearable devices; security vulnerabilities

1. Introduction

The Internet of Things (IoT) paradigm has encouraged the creation of environments with interconnected highly heterogeneous entities and networks. This can be reflected in the notable growth of wearable devices and applications over the last decade. Many of the devices are focused on monitoring physical activities for healthcare. That is the case of personal fitness trackers or wristbands, which offer new functionalities that require the collection of a wide amount of sensitive information about the user.

Fitness trackers designed to be worn all day long by users, track steps and physical activity of their owners, but they are also able to track their sleep duration and consistency, and many other factors related to users' health. Therefore, they are designed to locally gather all the information they can about the user throughout the day. The gathered information can later be synchronized with a mobile application, and then sent to servers controlled by fitness tracker manufacturers.

There is an intrinsic need to ensure users privacy and to minimize security vulnerabilities. For this reason, several studies [1–6] have showed the vulnerabilities and risks of these devices and security improvements have been carried out. Nevertheless, recent studies [7–9] have discovered still privacy and security issues, e.g., related to geolocation or unclear policies leaving the door open for the sale of users' fitness data to third parties without express consent of the users. In this study we analyse the potential security vulnerabilities of the communication of one of the most popular fitness trackers in the market[1] with the mobile application, the synchronization with the manufacturer's servers, and the

1 The fitness tracker model is not revealed to be compliant with the legal policy terms of service.

data shared with third party applications. The tracker -used in this study- has not been analysed in the literature and uses the same protocols that are used by most devices. In addition, the study has been carried out during several months, allowing us to discover privacy vulnerabilities that have not been analysed so far in the literature. The OAuth2 protocol implementation has been also analysed, as part of the traffic exchanged between the mobile application and the servers. To carry out the study, we have used Bluetooth test tools and sniffer software.

This paper is organized as follows. Section 2 describes the previous studies performed about vulnerabilities in personal fitness trackers and introduces Bluetooth Low Energy (BLE) as communication protocol between wearable and mobile devices. Section 3 gives details about the study performed. Section 4 describes the privacy and security vulnerabilities found in the communication between wearable and mobile app, the communication between mobile app and server and the API. Finally, Section 5 contains the conclusions, recommendations from the results and future work.

2. Related Work and Background

2.1. Related Work

One of the first contributions of security research on wearable devices is related to the popular Fitbit trackers and dates back to 2013. In this study, Rahman et al. reverse engineered the semantics of the tracker memory banks, the command types and the tracker-to-social network communication protocol [1]. In addition to finding vulnerabilities in the device, they also developed FitBite, a suite of tools that could allow an attacker to capture and modify data stored in one of these trackers, as well as Fitlock, a Fitbit extension that protected it against such attacks. This study was followed up by Zhou et al. that identified flaws in Fitlock and discussed possible means to alleviate these issues [10]. After that, Rahman et al. developed Garmax, similar to FitBite, as a tool to exploit vulnerabilities against Garmin trackers. Rahman et al. proposed SensCrypt, a protocol to protect low power fitness trackers (i.e., Fitbit and Garmin) [11].

Another study was carried out by Cyr et al., which tried to understand how much Fitbit had progressed in adding security to their devices as well as what data they collected, how they collected it and how it was transferred back and forth from the server. As a noticeable finding, they discovered that it was possible to obtain the BLE credentials sent to the phone from the server in plaintext [3].

In 2014, the security of Fitbit devices was put in the spotlight thanks to the strong media coverage of a security flaw detected by Axelle Apvrille [2]. This Fortinet researcher discovered a vulnerability that allowed an attacker to inject, over a Bluetooth connection, unauthorized code onto a Fitbit Flex and not only have it persist, but also reflect in the devices to which it connected. Other Fitbit model, the Charge HR, has been studied by Schellevis et al. [7]. They used reverse engineering of the cryptographic primitives used by the device and they were able to describe its authentication protocol. Among their findings, it is worth mentioning the discovery of a backdoor in the first firmware that enabled attackers to obtain via Bluetooth the encryption key stored in the tracker's memory.

Schiefer and Losche studied the security of data storage and transmission, either via Bluetooth or the Internet, and the authentication on application of nine fitness trackers (i.e., LG, Sony, Fitbit, Withings, Jawbone, Polar, Huawei, Garmin and Acer) [4]. For that, they developed a mobile application to capture all traffic sent to and received from the Internet, in order to search for unencrypted information. The trackers, except Sony and Huawei, use BLE to connect to the mobile device. The results show most devices do not allow deactivating Bluetooth, except Sony and Polar. All of them sent encrypted traffic, but the security level is different. This analysis was just a start up. Other studies, in the same year (i.e., 2015) and focused on the BLE communication, demonstrate the possibility of discovering and connecting to the devices without the owner even knowing [6] and how a theoretically robust protocol can be wrecked by badly implementing it, using a Nike+ tracker [5]. The last study found out the same vulnerability than [6], as well as the protocol supports direct reading and writing of the device memory (up to 65 K of contents).

Rieck demonstrates a flaw in the firmware for Withings' Activité, which allows an adversary to compromise the tracker itself. He suggests also findings that can be transferred to other trackers as well, due to the hardware similarities [12].

Despite the security improvements have been made on the fitness trackers after these studies, recent studies, such as [8,9], have proven that it is still necessary to keep on developing new security solutions. Hilts et al. analysed eight popular fitness trackers focusing on the privacy and security of data transmissions. They found security issues in most of the studied devices, and only one manufacturer implemented BLE privacy [8]. The authors in [9] presented a security analysis of a representative sample of current fitness trackers on the market. They focused on malicious user setting that aims at injecting false data into the cloud-based services leading to erroneous data analytics. Their results show the fitness device has no data integrity check and the collected data are stored in plain text on the smartphone.

Not only device vulnerabilities, but also application vulnerabilities may have a large impact in data disclosure. For instance, related to military information, in November 2017 the fitness tracking application Strava announced it will publish a heat map reflecting billions of tracked user activities [13]. Later, the 28th and 29th of January 2018, several media announced that Strava giving away the location of US military or intelligence operations, including secret military bases and patrol routes. It was first alerted by Nathan Ruser, a student of the Australian National University in a tweet.

A recent tendency is to share information and compete with friends on activity-based social networks [14]. This would raise even more security and privacy challenges, because an effective and granular access control over data must be appropriately provided.

The main contributions of this study beyond the literature are: discovery of new vulnerabilities related to privacy and denial of service (DoS) attacks in BLE communication; analysis of the traffic exchanged between mobile app and server, including how the OAuth2 protocol is used; and possible vulnerabilities identification in the mobile app-server communication.

2.2. *Bluetooth Low Energy (BLE)*

BLE, introduced as part of the Bluetooth 4.0 core specification [15,16], is a light-weight subset of classic Bluetooth. It was developed with the purpose of supporting devices that run for long periods on power sources, such as coin cell batteries or energy-harvesting devices. The BLE architecture is depicted in Figure 1, which shows the protocol stack in three components that contain layers or modules.

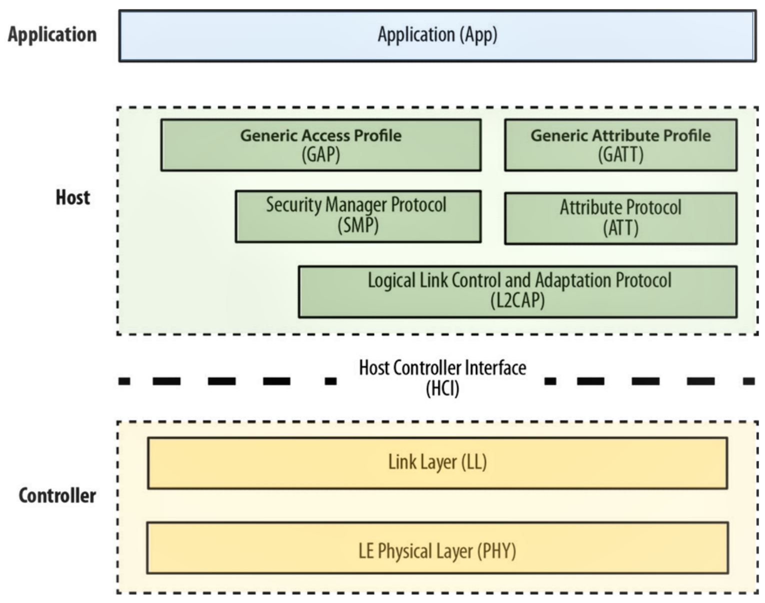

Application
Application (App)
Host
Generic Access Profile (GAP)
Generic Attribute Profile (GATT)
Security Manager Protocol (SMP)
Attribute Protocol (ATT)
Logical Link Control and Adaptation Protocol (L2CAP)
Host Controller Interface (HCI)
Controller
Link Layer (LL)
LE Physical Layer (PHY)

Figure 1. Bluetooth Low Energy Protocol Stack.

Controller. It includes the lowest layers of the BLE stack. The *Physical Layer* is in charge of modulating and demodulating analogous signals and transforming them into digital symbols. BLE devices can communicate using *unicast* or *broadcast* connections.

In a *unicast* connection, the device roles can vary depending on the mode: *non-connected* or *connected*. In *non-connected* mode, two main roles can be distinguished: *peripheral* and *central*. The *peripheral* role is usually played by a device constrained in both computing resources and energy, since its main tasks are to advertise and wait for connections. However, the second role, the *central*, has to be played by a device with more resources. It is the one responsible for scanning for other devices and it is also in charge of establishing links by sending connection requests.

Once a connection is established, i.e., *connected* mode, the *central* device can be referred to as *master*, while the *peripheral* is called *slave*. A slave can only be connected to a single master, but a master can be connected to multiple slaves.

On the other hand, regarding *broadcast* connections, the assigned roles do not change. The BLE device sending packets is called *broadcaster* while those that receive them are named *observers*.

The *Link Layer* (LL) interacts directly with the Physical Layer and it is responsible for advertising, scanning and creating or maintaining connections. This uses seven different advertising channel PDU (Protocol Data Unit) types: `ADV_IND` (connectable, undirected advertising), `ADV_DIRECT_IND` (connectable, directed advertising), `ADV_NONCONN_IND` (non-connectable undirected advertising), `ADV_SCAN_IND` (scannable undirected advertising), `SCAN_REQ` (active scanner), `SCAN_RSP` (scan response) and `CONNECT_REQ` (establishing a connection). It is also in charge of encryption, useful to ensure the confidentiality of the transmitted data, and filtering out advertising packets depending on their content or the Bluetooth Device Address (BDA), a 48-bit number that uniquely identifies a device.

Host. It involves protocols and profiles. The profiles use functionalities provided by the underlying protocols.

The *Generic Access Profile* (GAP) modes and procedures can be seen as the cornerstone of BLE. The GAP defines how BLE devices can communicate with each other. It is responsible for the management of the three mechanisms that BLE devices employ to communicate: discovery, connecting and **pairing**. GAP is also in charge of securing communications, by making use of rules and algorithms implemented in the *Security Manager Protocol*. After two devices have established a connection, their communication does not initially have security, so they must pair. The pairing process entails authenticating the identity of the master and the slave, encrypting the link with a short-term key (STK) and, afterwards, distributing long-term keys (LTK). The devices generate an ECDH (*Elliptic Curve Diffie-Hellman*) public-private key pair to, right after, exchange their public keys and start computing the Diffie-Hellman key.

The *Generic Attribute Profile* (GATT) determines the way data are organized and sent over a BLE connection. It makes use of the Attribute Protocol (ATT) as transport mechanism, as well as organizing data into attributes, making its transmission much easier.

These profiles and protocols use the *Logical Link Control and Adaptation Protocol* (L2CAP) to pass packets to either the *Host Controller Interface* (HCI) or on a hostless system, directly to the LL.

Application. At the top of architecture, the *Application* represents apps that can use the functionalities of the BLE Protocol Stack by accessing the API.

3. Study Description and Setup

The vulnerability study has been mainly focused on the possible privacy and security vulnerabilities of the communications among agents: wearable device, mobile application and server. So the study has been divided in two main sections: the communication between wearable device (i.e., fitness tracker) and mobile application (i.e., in Android, iOS and others) using BLE; and communication between mobile application and server, using Internet. In addition, the manufacturer offers an API service for third party applications to interact with the user's data. The API offers a registry process that involves OAuth 2.0 protocol [17,18].

The communication between the tracker and the mobile application is handled by BLE, also referred to as *Bluetooth Smart*. For sniffing the BLE communication, a passive analysis was carried out by employing two different methods:

1. enabling the *Bluetooth HCI snoop log* Developer Options feature in Android phone. When this feature is enabled, a log file (`btsnoop_hci.log`) is created. In this file, all the BLE HCI packets (i.e., packets exchanged between controller and the host interface) are registered.
2. using an Ubertooth-One hardware and software (in a laptop) to monitor and decode Bluetooth packets.

The resulting dump file in the Android phone and the traffic captured by Ubertooth were inspected by using Wireshark. In addition, we use mobile apps such as *BLEScanner* and *nRF Connect*.

The communication between the mobile application and the server (e.g., cloud-based services) uses HTTP/HTTPS and the OAuth 2.0 protocol for authorization. Applications to capture the traffic were used such as *Inspeckage* and *mitmproxy*. These same tools were used for testing third party applications. In addition, we make reverse engineering in the mobile app in order to bypass the certificate pinning.

The whole scenario is depicted in Figure 2, where different agents interacting and a laptop with the sniffer tools and also the Bluetooth hardware can be seen.

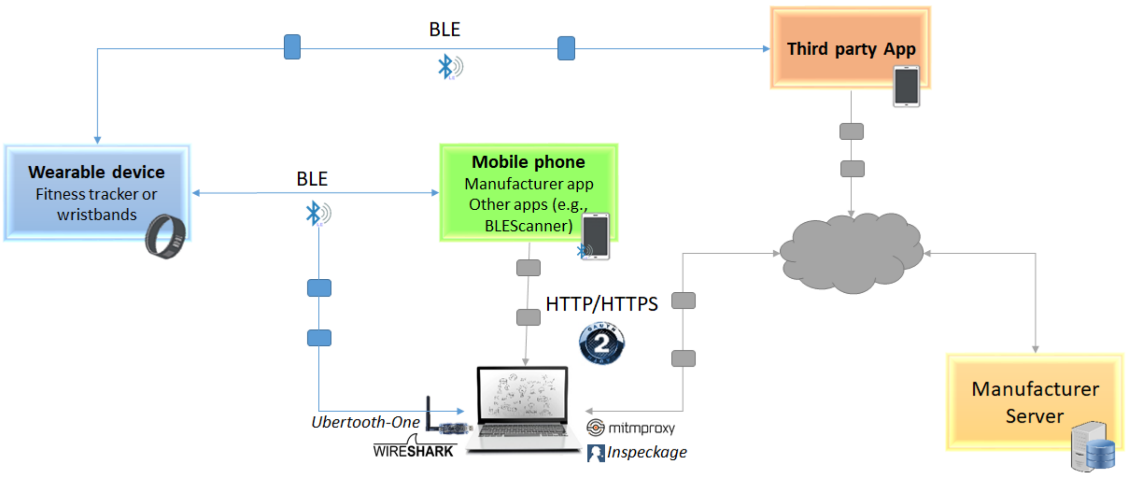


Figure 2. Scenario for communication testing.

The study was carried out during four months approximately. In those months, we performed different tests to assess the security of the tracker, mobile app and server communications and API. The Table 1 shows the tests carried out in: the communication between wearable and mobile devices (Section 4.1), the communication between mobile app and server (Section 4.2) and the API (Section 4.3).

Table 1. Tests carried out for each part of the study.

Study Part	Tests
Wearable device—mobile application communication	**T1:** Pairing device with Fitbit app **T2:** Pairing device with third party app **T3:** Syncing with device
Mobile application—server communication	**T4:** Signing up for app **T5:** Log out of app **T6:** Logging into app **T7:** Editing profi le data **T8:** Logging activities manually **T9:** Deleting a device **T10:** Pairing device with server app **T11:** Syncing with device
Third party API	**T12:** Signing up for app **T13:** Logging into app **T14:** Editing profile le data **T15:** Logging activities

The data managed by this kind of devices are related to the user, such as personal, contact and physical activity, and to the device. Table 2 summarizes the attributes that can be managed, grouped by basic and additional attributes. "Basic attributes" are the data given by the user when creating an account and registering his/her device, information collected by the tracker and the technical information about device when it synchronizes with the mobile application. "Additional attributes" allow users to provide more information in order to enjoy additional services (e.g., associating a Facebook or Google+ account, sending messages and logging food) or customize his/her account (e.g., profile photos).

Table 2. Data managed by fitness trackers.

Data type	Basic Attributes	Additional Attributes
Personal data (PD)	Gender Age Date of birth Height Weight	Food log Water log Location Profile photos Alarms
Contact info (CI)	email	Contacts Basic info from other accounts (e.g., social networks)
Activity data (AD)	Steps Floors Heart rate Sleep quality	
Device info (DI)	Battery level Sync time App info	

These data represent the assets to be protected, because these are shared by all agents in the scenario. The data can be grouped in three categories according to the sensitivity of the information: PD, AD and DI. PD contains the personal data and contact information that allow identifying the users, called *Personally Identifiable Information*. AD contains the user's physical activity data gathered by the wearable device. With the aggregated activity data would be more difficult to identify a user (than using PD). DI contains information about the tracker or the user's mobile phone, including location.

4. Vulnerabilities Results

4.1. Vulnerabilities in the Communication between Wearable and Mobile Devices

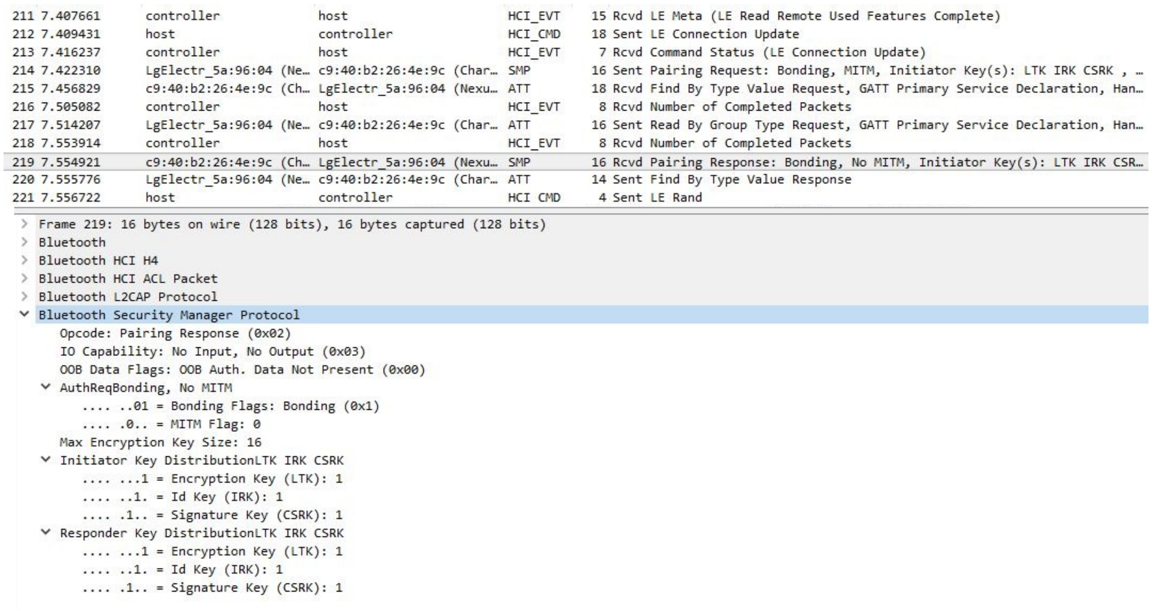
After analysing the pairing and synchronization processes (i.e., T1 to T3 in Table 1) between wearable and mobile devices, using manufacturer mobile app and free apps to manage BLE connections, the more relevant possible vulnerabilities found were:

- **Use of Public BDAs and General Discovery.** After keeping track of the fitness tracker's BDA, it was found that its value did not change, which involves that it uses a Public BDA. Given its value is fixed and unique, a user could be tracked by this identifier, violating her/his privacy, because this information is contained in all the advertising PDUs.

  In addition, the risk increases if we take into account the release of applications such as RamBLE that scan for BLE devices in order to obtain information from their undirected advertising packets (`ADV_IND`) and their `SCAN_RSP`. Besides, the application also stores the device's BDA with the geographic location at which it was detected. The information is shown graphically in a map. This information can be obtained easily, because the tracker always uses general discovery instead of direct discovery, even after being bonded with the mobile phone, so the `ADV_IND` packets are always sent.

  For this reason, devices send advertising packets and respond to scanning requests. Thus, it is also possible to develop an app that sends `SCAN_REQ` messages and the wearable device answers with a `SCAN_RESP`. The response contains information about device: its name and the current transmission power. This information may be used to plan a denial of service attack.

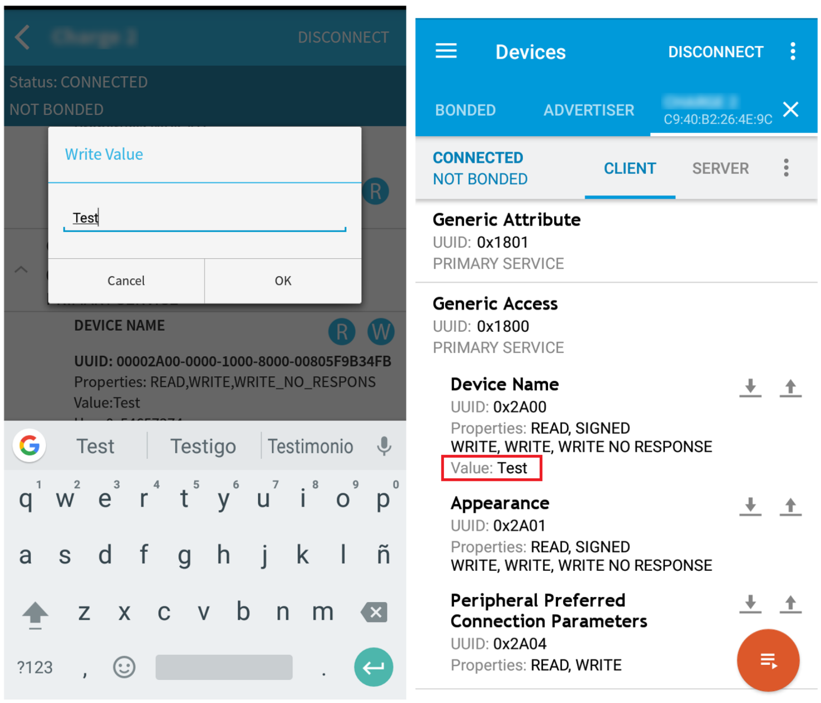
  This weakness is emphasized by the fact that the tracker always sends undirected advertising instead of direct discovery (i.e., `ADV_DIRECT_IND`), even after being bonded with a mobile phone. So devices are constantly sending advertising packets and responding to scanning requests. In this sense, a DoS attack can be led.
- **Not use of BLE Secure Connections.** Sniffing the pairing of the application and the tracker, we can observe that the out of band (OOB) method is used to set the keys. This is one of the legacy methods, Secure Simple Pairing model, and it is not using the long term key derivation using ECDH as proposed by BLE security specification. Figure 3 shows the capture of the bonding negotiation where the OOB method is identified. The key is generated only once, in this initial pairing and stored in the bonding with the device. Despite the new BLE security specification is not followed, we were able to see its own encryption and integrity (24-bit CRC) to protect the Bluetooth connection.
- **Connections with unbounded devices.** We also found the device accepts connections from non-bonded devices, similar to results of other studies. This process leaks several details about services and UUIDs, and BDAs. Armed with the UUID of a device, an attacker can monitor the tracker even if a private BDA was used. Even worse, we found that using a Bluetooth scanner application, e.g., *BLEScanner* or *nRF Connect*, we were able to modify the value of some attributes, for instance the tracker name. This can at least confuse the user (see Figure 4).



```
211 7.407661    controller            host                  HCI_EVT    15 Rcvd LE Meta (LE Read Remote Used Features Complete)
212 7.409431    host                  controller            HCI_CMD    18 Sent LE Connection Update
213 7.416237    controller            host                  HCI_EVT     7 Rcvd Command Status (LE Connection Update)
214 7.422310    LgElectr_5a:96:04 (Ne… c9:40:b2:26:4e:9c (Char… SMP  16 Sent Pairing Request: Bonding, MITM, Initiator Key(s): LTK IRK CSRK , …
215 7.456829    c9:40:b2:26:4e:9c (Ch… LgElectr_5a:96:04 (Nexu… ATT  18 Rcvd Find By Type Value Request, GATT Primary Service Declaration, Han…
216 7.505082    controller            host                  HCI_EVT     8 Rcvd Number of Completed Packets
217 7.514207    LgElectr_5a:96:04 (Ne… c9:40:b2:26:4e:9c (Char… ATT  16 Sent Read By Group Type Request, GATT Primary Service Declaration, Han…
218 7.553914    controller            host                  HCI_EVT     8 Rcvd Number of Completed Packets
219 7.554921    c9:40:b2:26:4e:9c (Ch… LgElectr_5a:96:04 (Nexu… SMP  16 Rcvd Pairing Response: Bonding, No MITM, Initiator Key(s): LTK IRK CSR…
220 7.555776    LgElectr_5a:96:04 (Ne… c9:40:b2:26:4e:9c (Char… ATT  14 Sent Find By Type Value Response
221 7.556722    host                  controller            HCI CMD     4 Sent LE Rand

> Frame 219: 16 bytes on wire (128 bits), 16 bytes captured (128 bits)
> Bluetooth
> Bluetooth HCI H4
> Bluetooth HCI ACL Packet
> Bluetooth L2CAP Protocol
v Bluetooth Security Manager Protocol
    Opcode: Pairing Response (0x02)
    IO Capability: No Input, No Output (0x03)
    OOB Data Flags: OOB Auth. Data Not Present (0x00)
  v AuthReqBonding, No MITM
      .... ..01 = Bonding Flags: Bonding (0x1)
      .... .0.. = MITM Flag: 0
    Max Encryption Key Size: 16
  v Initiator Key DistributionLTK IRK CSRK
      .... ...1 = Encryption Key (LTK): 1
      .... ..1. = Id Key (IRK): 1
      .... .1.. = Signature Key (CSRK): 1
  v Responder Key DistributionLTK IRK CSRK
      .... ...1 = Encryption Key (LTK): 1
      .... ..1. = Id Key (IRK): 1
      .... .1.. = Signature Key (CSRK): 1
```

Figure 3. Bonding Negotiation.



[redacted] DISCONNECT
Status: CONNECTED
NOT BONDED
Write Value
Test
Cancel OK
DEVICE NAME
UUID: 00002A00-0000-1000-8000-00805F9B34FB
Properties: READ,WRITE,WRITE_NO_RESPONS
Value:Test

Devices DISCONNECT
BONDED ADVERTISER [redacted] C9:40:B2:26:4E:9C
CONNECTED
NOT BONDED
CLIENT SERVER

Generic Attribute
UUID: 0x1801
PRIMARY SERVICE

Generic Access
UUID: 0x1800
PRIMARY SERVICE

Device Name
UUID: 0x2A00
Properties: READ, SIGNED
WRITE, WRITE, WRITE NO RESPONSE
Value: Test

Appearance
UUID: 0x2A01
Properties: READ, SIGNED
WRITE, WRITE, WRITE NO RESPONSE

Peripheral Preferred
Connection Parameters
UUID: 0x2A04
Properties: READ, WRITE

Figure 4. Non-Bonded but connected device altering attributes.

All of these possible vulnerabilities are related to the assets PD and DI: users' location and device information.

4.2. *Vulnerabilities in the Communication between Mobile Devices and Server*

The mobile device runs an app which communicates with the server. The application plays the role of forward proxy. In this part of the study, eight tests related with signing, logging, edition, pairing and synchronization processes were carried out (i.e., T4 to T11 in Table 1).

In order to inspect the security of this communication we used a proxy to inspect the traffic between them. The proxy was configured to intercept and impersonate the server to the client, issuing certificates on the fly according to the incoming requests from the devices. The certificates have to be issued according to the expected `CommonName`, the `AltName`, and the `SNI` (Server Name Indication) requested by the client at the initiation of the connection.

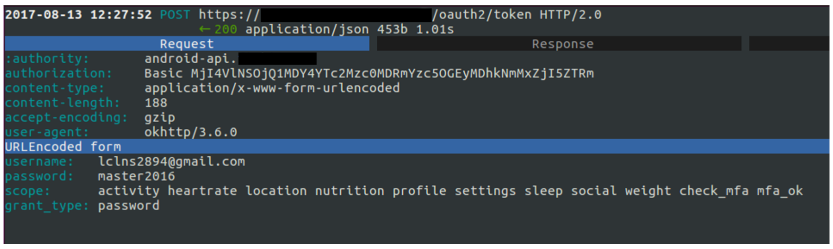
The CA (*Certification Authority*) certificate used by the proxy to dynamically issue the certificates has to be installed in the mobile device. This should be enough to have access to the clear text communications of the app with the server. Luckily, certificate pinning has been deployed in the app. This extra security measure allows the app to refuse a secure connection when the certificate passed by the server does not match the known certificate by the app.

To bypass the certificate pinning, the app has to be altered to remove the certificate pinning support. We download and disassemble the apk, locate the java methods where the certificate pinning is used. Inserting a `return void` statement at the beginning of the method suffices to bypass the certificate checking and thus the certificate pinning. We repackage the app and install it locally in the mobile device. Due to the mechanism of verification of certificates in Android when apps are installed, it is relatively easy to reassemble an app (.apk file), even using other certificates (i.e., fake certificates) different from developer.

At this point, we have **access to the clear text communications** between the mobile device and the server. We start the app and inspect the exchanged traffic for vulnerabilities and security issues.

- **Authentication credentials sent in cleartext.** In Figure 5 we observe that the app uses HTTP Basic authentication to get an OAuth authentication token. This means the app is sending the email and password in cleartext to the server. Even though the connection is protected by TLS, the password should not be sent in cleartext. At minimum it should be hashed together with a nonce freshly generated for each connection to avoid pass the hash attacks. This fact compromises the assets PD, AD and DI.
- **Misuse of OAuth2.** The app also makes *use of bearer tokens* without protection mechanisms (e.g., digital signature, hashes, or others). OAuth mechanism does not require proof of possession of any key. The problem is that these tokens are extensively used by the apps without integrity means and they are valid for 8 hours, though according to RFC 6750 [19] they should be protected with a MAC (Message Authentication Code) and the maximum validity should be 1 h.

  Besides, once the tracker has been setup, the app is used to forward tracking information to the server. We have changed for instance the parameter `btlename` and the server does not complain. The Bluetooth device address and the ID of the user are sent encoded and kept constant, even after app reinstallation, making easier to track a particular user. This lack of privacy is worse if we look at the `stats` sent data such as user's phone Android ID or the Wi-Fi SSID. These data can compromise the user privacy, and should not be sent to the servers.
- **Unauthorized data share.** Regarding data shared with third parties, together with data related to the tracker, the app also sends data regarding the mobile device (e.g., manufacturer, OS version, screen, paired devices and so on). This again should be explicitly authorized by the user. We must also say that the ID sent for these purposes is a different one from the ID used within the manufacturer servers.



```
2017-08-13 12:27:52 POST https://[redacted]/oauth2/token HTTP/2.0
                         ← 200 application/json 453b 1.01s
                  Request                                   Response
:authority:          android-api.[redacted]
authorization:       Basic MjI4VlNSOjQ1MDY4YTc2Mzc0MDRmYzc5OGEyMDhkNmMxZjI5ZTRm
content-type:        application/x-www-form-urlencoded
content-length:      188
accept-encoding:     gzip
user-agent:          okhttp/3.6.0
URLEncoded form
username:   lclns2894@gmail.com
password:   master2016
scope:      activity heartrate location nutrition profile settings sleep social weight check_mfa mfa_ok
grant_type: password
```

Figure 5. Requesting OAuth token with cleartext passwd.

4.3. Vulnerabilities in the Third Party API

Finally, we developed an application to test the public API for third party applications. In this part, we carried out four tests related with signing, logging and edition (i.e., T12 to T15 in Table 1).

We found some insecurities that may be improved, such as allowing the use of `WebView`, which does not offer the same level of protection as normal browsers, and can be easily faked. The API only supports access tokens but lacks support for OAuth Refresh tokens. Finally, we have detected that callback URLs do not force HTTPS, allowing MiTM attacks.

5. Conclusions and Future Work

This study shows that manufacturers have keep improving the design of their tracker devices. In spite of several improvements being introduced in security at Bluetooth Low Energy, they have not yet implemented them. It conveys possible vulnerabilities related to privacy and security. For many companies, security is only present at the latest phases of the design and during the maintenance period of devices, when they have to react to vulnerabilities and related issues. We expect the new EU Regulations on Data Protection and Privacy will help manufacturers understand the importance of security and introduce security in the whole design cycle, from the beginning of the design, to the end of the supported life. Some recommendations to correct the found main issues are:

- Regarding the tracker-application communication, public BDAs are used. They should be replaced by private BDAs, or frequently changing addresses. The device should establish connections with trusted devices, not accepting connections with unbounded devices, nor making it possible the modification of some attribute values like the device name. Besides once the device is bonded with a trusted device, the tracker should only use Bluetooth Direct Discovery. General discovery brings no advantage here, only snooping the Bluetooth devices in the neighbourhood, which is not something meaningful for the purpose of the tracker, and for user side, no user data should be transmitted to the manufacturer servers.
- Regarding the application-server communication, the bearer tokens should be adjusted to an hour or less, according to the standard. The use of cleartext passwords should be avoided and replaced by some of the many available schemas. At least, a hash including a session nonce. We could not assess the internal security measures of the manufacturer servers, but cleartext password storage must be avoided. Also the hardware identifiers should be avoided at this level of communication (tracker BDA), and explicit user consent is needed for each specific type of information that the app is transmitting (e.g., user Android ID, Wi-Fi SSID, observed Bluetooth connections). This data makes easier profiling users, and may get more cyber-criminal attention.
- Finally, the manufacturer should improve the analysis of third party applications, effectively avoiding the access with WebView, banning callbacks to HTTP URLs, and implementing OAuth refresh tokens where the third party should be authorized every 30 days at maximum.

We are going to perform the study in other fitness trackers and to analyse privacy and security issues related to the use of data by third party applications and activity-based social networks.

Funding: This work was supported by the Spanish Ministry of Science and Innovation (MINECO) through the project INRlSCO: Incident Monitoring in Smart Communities (TEC2014-54335-C4-2-R) and MAGOS: Inteligencia de fuentes abiertas para "Smart Grids" (TEC2017-84197-C4-1-R).

References

1. Rahman, M.; Carbunar, B.; Banik, M. Fit and Vulnerable: Attacks and Defenses for a Health Monitoring Device. *arXiv* **2013**, arXiv:1304.5672.
2. Apvrille, A. Geek usages for your Fitbit Flex tracker. In Proceedings of the Hack.lu Conference, Luxembourg, 20–22 October 2015.
3. Cyr, B.; Horn, W.; Miao, D.; Specter, M. *Security Analysis of Wearable Fitness Trackers (Fitbit)*; Technical Report; Massachusetts Institute of Technology (MIT): Cambridge, MA, USA, 2013.
4. Clausing, E., Schiefer, M.; Lösche, U. *Internet of Things Security Evaluation of nine Fitness Trackers*; Technical Report; AV TEST: The Independent IT-Security Institute: Magdeburg, Germany, 2015.
5. Margaritelli, S. *Nike+ FuelBand SE BLE Protocol Reversed*; Technical Report; Evilsocket.net. 2015.
6. Unucheck, R. *How I Hacked My Smart Bracelet*; Technical Report; Kaspersky Lab: Moscow, Russia, 2015.
7. Schellevis, M.; Jacobs, B.; Meijer, C.; de Ruiter, J. *Getting Access to your Own Fitbit Data*; Radboud University: Nijmegen, The Netherlands, 2016.
8. Hilts, A.; Parsons, C.; Knockel, J. *Every Step You Fake: A Comparative Analysis of Fitness Tracker Privacy and Security*; Technical Report; Open Effect: Toronto, ON, Canada, 2016.
9. Fereidooni, H.; Frassetto, T.; Miettinen, M.; Sadeghi, A.R.; Conti, M. Fitness Trackers: Fit for Health but Unfit for Security and Privacy. In Proceedings of the IEEE/ACM International Conference on Connected Health: Applications, Systems and Engineering Technologies (CHASE), Philadelphia, PA, USA, 17–19 July 2017; pp. 19–24. doi:10.1109/CHASE.2017.54.
10. Zhou, W.; Piramuthu, S. Security/Privacy of Wearable Fitness Tracking IoT Devices. In Proceedings of the 9th Iberian Conference on Information Systems and Technologies (CISTI), Barcelona, Spain, 18–21 June 2014. doi:10.1109/CISTI.2014.6877073.
11. Rahman, M.; Carbunar, B.; Topkara, U. Senscrypt: A secure protocol for managing low power fitness trackers. In Proceedings of the IEEE 22nd International Conference on Network Protocols, Raleigh, NC, USA, 21–24 Oct 2014; pp. 191–196.
12. Rieck, J. Attacks on Fitness Trackers Revisited: A Case-Study of Unfit Firmware Security. *arXiv* **2016**, arXiv:1604.03313.
13. Hsu, J. The Strava Heat Map and the End of Secrets, Security news, Jan 2018. Available online: www.wired.com.
14. Pham, A.; Huguenin, K.; Bilogrevic, I.; Dacosta, I.; Hubaux, J.P. SecureRun: Cheat-Proof and Private Summaries for Location-Based Activities. *IEEE Trans. Mob. Comput.* **2016**, *15*, 2109–2123.
15. Bluetooth SIG (Special Interest Group). *Bluetooth Core Specification v4.0*, 2009. Available online: www.bluetooth.com.
16. Bluetooth SIG (Special Interest Group). *Bluetooth Core Specification v5.0*, 2016. Available online: www.bluetooth.com.
17. Hardt, D. *The OAuth 2.0 Authorization Framework*; RFC6749; Internet Engineering Task Force (IETF): Fremont, CA, USA, October 2012.
18. Denniss, W.; Bradley, J. *OAuth 2.0 for Native Apps*; RFC8252; Internet Engineering Task Force (IETF): Fremont, CA, USA, October 2017.
19. Jones, M.; Hardt, D. *The OAuth 2.0 Authorization Framework: Bearer Token Usage*; RFC6750; Internet Engineering Task Force (IETF): Fremont, CA, USA, October 2012.