Internet of Things

Report by Chiara Simoni and
Giada Martina Stivala

Some project on debugging Bluetooth Low Energy devices

A.A. 2017/2018
Prof. Luciano Bononi, Marco Di Felice

July 22, 2018

Contents

Chapter 1

Introduction

Bluetooth Low Energy devices are flooding the market of cheap, wearable, health, home-appliance electronics. Some examples include monitoring devices for sport, health and outdoor activities in general; they also connect many household appliances as well as security cameras and locks. Other interesting use-cases include wristbands used in theme parks or live music concerts (although the technology may differ).

Intrinsically, these devices perform very simple tasks, mostly in the field of data gathering. It seems totally harmless to have a smart watch tracking your position at any time of day, but what if it were possible for anyone to query it and extract its data? Their minimalism is what makes them so appealing to the general public, but it is sometimes their intrinsic weakness.

Moving a bit further from *"hackers"* and information security, data gathering is a fundamental step for data analysis in a Big Data context. Privacy concerns should be taken into consideration also in terms of data anonymization.

In this project report, we will start by describing general device characteristics and Bluetooth specifications. We will then move to their concrete observation with hands-on analyses, taking into consideration different developing tools and devices. In particular, we will focus our attention on security risks and concrete flaws, highlighting security exploits when possible. Finally, we will summarize our findings, including a few ideas for future directions and developments.

For our tests, we used a Mi Band 2 smart band, a Magic Blue smart lightbulb and a ST Microelectronics IoT node (model `STM32L475 MCU`). The code was run both on Ubuntu 16.04 and Kali Linux 4.13. It is worth mentioning that we also tried to use a Ubuntu GNOME OS, virtualized via VMWare on Windows 10, but this option was soon discarded as VMWare does not provide the drivers for Bluetooth Low Energy.

The software at our disposal included the builtin Linux commands, as well as Wireshark and Android apps (namely BLEScanner and nRFCon-

nect). As we will see, we also tested some open-source tools available from Github: bleah, btlejuice and gattacker.

Chapter 2

Bluetooth Low Energy Specification

In this chapter we briefly report the main technical characteristics of the Bluetooth Low Energy Personal-Area-Network technology. Our main reference is the Core Specification 5.0 available on the official website.

2.1 Network Topology and Devices

Devices can be divided into three categories: Advertisers, Scanners and Initiators. Communication has to be started by an initiator device following an *advertisement connectable packet* (`ADV_IND`). Advertisement is performed on the primary channels, while bidirectional communication happens on one of the 37 secondary channels, decided during the connection procedure. When a connection is established, the initiator and the advertiser respectively become the master and slave devices; while the former can be involved in more than one communication, the slave device can belong to a single *piconet* at a time. In fact, it is not possible to have a BLE device paired with more than one "master" device at the same time.

Each device's role is defined by the Generic Access Profile (GAP). As such, we can say that GAP contributes in shaping the topology of the network. Specifically, GAP defines the *Broadcaster*, *Observer*, *Peripheral* and *Central* roles. Broadcaster and Observer devices are, respectively, optimized for broadcast and reception; differently, Peripheral and Central devices work better in bidirectional communications.

As we had the possibility to concretely program a BLE device, we show in Listing 2.1 an extract of code in which the device is advertising *connectable* packets.

Listing 2.1: Set up flags for device mode of operation: connectable

```
1 ble.gap().accumulateAdvertisingPayload(GapAdvertisingData::
    BREDR_NOT_SUPPORTED | GapAdvertisingData::
```

```
    LE_GENERAL_DISCOVERABLE);
ble.gap().accumulateAdvertisingPayload(GapAdvertisingData::
    COMPLETE_LIST_16BIT_SERVICE_IDS, (uint8_t *)uuid16_list,
    sizeof(uuid16_list));
ble.gap().accumulateAdvertisingPayload(GapAdvertisingData::
    THERMOMETER_EAR);
ble.gap().accumulateAdvertisingPayload(GapAdvertisingData::
    COMPLETE_LOCAL_NAME, (uint8_t *)DEVICE_NAME, sizeof(
    DEVICE_NAME));
ble.gap().setAdvertisingType(GapAdvertisingParams::
    ADV_CONNECTABLE_UNDIRECTED);
ble.gap().setAdvertisingInterval(1000); /* 1000ms. */
ble.gap().startAdvertising();
```

Advertising may also be *non-connectable* (`ADV_NONCONN_IND`): the LE device periodically sends its data on the main channel for every scanner to read. This is really convenient in terms of implementation, but it also represent the least secure solution, as data is sent in clear text. Moreover, since there is no acknowledgment nor response by the receiving party, this method is also the less reliable. In Listing 2.2 we highlight the different parameter used to set *non-connectable* advertising.

Listing 2.2: Set up flags for device mode of operation: non connectable

```
ble.gap().setAdvertisingType(GapAdvertisingParams::
    ADV_NON_CONNECTABLE_UNDIRECTED);
```

On the whole, the Specification defines six different advertising packets, among which we also report the request for additional information from the advertisement (`SCAN_REQ`), it having a direct corresponding bash command. Advertisement is performed at random intervals to avoid collisions, thus it may happen that the target device is not immediately discovered by the scanner.

2.2 Connection

When the Scanner device answers to an advertisement message, a secondary channel is established for bidirectional communication. This channel is randomly chosen and communicated to the advertiser in the connection request packet. As previously mentioned, the slave device can be paired with a single master at a time, consequently all subsequent communications will happen in the established secondary channel, also granting additional connection speed.

Pairing procedures vary depending on the Bluetooth Low Energy device. They are mentioned in the Specification as *Association modes* and they are the following:

- Numeric Comparison: used when the devices can show numbers (at least 6 digits) and can receive user input (e.g. yes or no), like a phone

or a laptop. Devices wishing to pair should show the same number, and the owners have to confirm the match.

  - This usually happen when pairing two mobile phones or a mobile phone to a laptop. In case the BLE device has a display, this association mode is attempted.

- Just Works: used when 1+ devices doesn't have a keyboard for user IO and / or cannot display 6 digits. Consequently, no PIN is shown and the user just has to accept the connection.

  - This is the case of the STM IoT Node. When attempting to connect from any Scanner device, the connection is established without authentication.
  - It is also the case of the Magic Blue smart bulb: the device pairs with any scanner.
  - In both situations, to unpair from the device it is necessary to unplug from the power source.

- Out Of Band: used when there is a (secure) OOB channel for pairing and security keys exchange. Of course, if such channel is not secure the whole process may be compromised.

- Passkey Entry: one device has input capabilities, but can't display 6 digits; the other device has output capabilities: the PIN is showed on the second device, and the connection is "confirmed" by entering the same PIN on the first device. Note that in this case the PIN is created by a specific security algorithm, while in legacy versions was an input from the user.

  - This is the case of the Mi Band 2 smart band, in which it is not possible to display PIN numbers, but there is an input device: when attempting to pair, the band vibrates and asks for confirmation by tapping its button.

2.3 GATT Transactions

While GAP defines how BLE-enabled devices can make themselves available, GATT (Generic ATTribute Profile) defines in detail how two Bluetooth Low Energy devices can connect and transfer data back and forth. The communication features *Profiles*, *Services* and *Characteristics*: they make use of the Attribute Protocol (ATT) that stores all the details related to the device in a simple lookup table, using 16-bit IDs for each entry. Once the advertising process governed by GAP has concluded, it's the GATT turn to enter the scene. The established connection is not symmetrical: the Peripheral

can connect to a single Central while the Central can connect with multiple Peripheral devices. A bidirectional connection is the only way to share data between devices, although it may also be possible for Peripherals to exchange data between themselves through a mailbox system; this architecture has a central unit for message dispatch, but has to be manually implemented.

GATT basically supports a server/client relationship where the entities are called GATT Server and GATT Client, the latter being usually a phone or a tablet that sendings requests to the server. The master device is responsible for the initiation of the transaction, and once the connection is stable, the Peripheral will suggest a *Connection Interval* for the connection, to see if there is new data available. However it is not compulsory for the central device to honour the request of the client if resources are not available.

——–> INSERT IMAGE OF MASTER/SERVANT TRANSACTION HERE <——-

2.3.1 Profiles, Services and Characteristics

As previously mentioned, GATT transactions make use of high-level nested objects called Profiles, Services and Characteristics.

- Profile: it is the collection of Services on the device that has been compiled by the peripheral designers.
- Services: they contain specific chunks of data that are the Characteristics. A service can have one or more characteristics and each service is identified through a unique number called UUID, that can be either 16-bit or 128-bit depending on the manufacturing of the device. Services can be explored on the Services of the Bluetooth Developer Portal, thus it's easy to track a service and its purpose in case the Service is named "Unknown".
- Characteristics: this is the lowest level in a GATT transaction and contains a single data point (it may be a single value or an array of bytes, depending on the type of data transmitted). Characteristics are composed by various elements such as a type, a value, properties and permissions. Properties, in particular, define what another device can do with the characteristics over Bluetooth in terms of operations such as READ, WRITE or NOTIFY.
  - READING a characteristic means copying the current value to the connected device.
  - WRITING allows the connected device to insert new values of data.
  - NOTIFICATIONS consist in a message type periodically sent in case of changes in a value.

Permissions specify the condition that must be met before reading or writing data to the characteristic is granted.

- Descriptors: there is another level below the Characteristics level, that contains meta-data related to it. For example it's possible enable or disable notifications through the Client Characteristic Configuration Descriptor.

Bibliography

Appendix A

GATT Assigned Numbers

Data Type	Data Type Name	See advlib section
0x01	Flags	Flags
0x02	Incomplete List of 16-bit UUIDs	UUID
0x03	Complete List of 16-bit UUIDs	UUID
0x04	Incomplete List of 32-bit UUIDs	UUID
0x05	Complete List of 32-bit UUIDs	UUID
0x06	Incomplete List of 128-bit UUIDs	UUID
0x07	Complete List of 128-bit UUIDs	UUID
0x08	Shortened Local Name	Local Name
0x09	Complete Local Name	Local Name
0x0a	Tx Power Level	Tx Power
0x0d	Class of Device	Generic Data
0x0e	Simple Pairing Hash C-192	Generic Data
0x0f	Simple Pairing Randomizer R-192	Generic Data
0x10	Security Manager TK Value	Generic Data
0x11	Security Manager OOB Flags	Generic Data
0x12	Slave Connection Interval Range	SCIR
0x14	16-bit Solicitation UUIDs	Solicitation
0x15	128-bit Solicitation UUIDs	Solicitation
0x16	Service Data 16-bit UUID	Service Data
0x17	Public Target Address	Generic Data
0x18	Random Target Address	Generic Data
0x19	Public Target Address	Generic Data
0x1a	Advertising Interval	Generic Data
0x1b	LE Bluetooth Device Address	Generic Data
0x1c	LE Bluetooth Role	Generic Data
0x1d	Simple Pairing Hash C-256	Generic Data
0x1e	Simple Pairing Hash Randomizer C-256	Generic Data
0x1f	32-bit Solicitation UUIDs	Solicitation
0x20	Service Data 32-bit UUID	Service Data
0x21	Service Data 128-bit UUID	Service Data
0x22	LE Secure Con. Confirmation Value	Generic Data
0x23	LE Secure Connections Random Value	Generic Data
0x24	URI	Generic Data
0x25	Indoor Positioning	Generic Data
0x26	Transport Discovery Data	Generic Data
0x27	LE Supported Features	Generic Data
0x28	Channel Map Update Indication	Generic Data
0x29	PB-ADV	Generic Data
0x2a	Mesh Message	Generic Data
0x2b	Mesh Beacon	Generic Data
0x3d	3-D Information Data	Generic Data