Reverse engineering in communication (BLE) between a Smartwatch and a Smartphone

1st Abel Chils Trabanco

I. Introduction

Nowadays, in e-commerce companies, you can find a vast catalog of cheap smart devices. But often, the software provided to control them are a bit neglected. Although, they do not usually have a standardized way of communication with the smartphone and are only compatible with the official application.

In this work, it will be analyzed and documented how the communication protocol of one of these gadgets (Smartwatch F2 manufactured by DT N0.1) work [1]. Also, it will be created a Python tool that interact with it [2] and a mobile application that can substitute the original application [3].

During all the processes, the original mobile application won't be modified and only will be done observations of its behaviors, specifically the data they exchange with the gadget. This form of reverse engineering is covered by a European directive, EU Directive 2009/24/EC Article 5 in paragraph 3 [5]. Besides, there is a court sentence that endorses this type of study, specifically the case C-406/10 of the European court in paragraph 61 [6].

II. Background

Currently, many projects perform reverse engineering on gadgets. This project is inspired by an article in which is discovered the communication protocol of the NO.1 F4 model of the same brand. Unlike this work, where the system logs are used to obtain the packets that have been transmitted, in that article a tool named Gattacker [10] was used to perform a man-in-the-middle attack to discover the packages that have been transmitted. Although this approach allows for the identification of packages in a faster way, the tool was not able to establish communication with the Smartwatch used at this work, so the approach was discarded. Also, the article limits itself to discovering the communication protocol for a few functionalities and creating some Python scripts to test them. While in this work the communication protocol of all functionalities is cataloged and are developed a framework that allows control of the clock from the PC implementing all the functionalities and an Android application that can replace the original application.

On the other hand, some tools used in that article are used in this work too (e.g PyGatt library [11]). Besides, although the communication protocols are different, they share some similarities (e.g neither of the two protocols it encrypts the messages and both protocols use a checksum).

III. System description

The Smartwatch to be analyzed names F2 and is manufactured by DT N0.1 [8]. It has a Bluetooth V4.0 transmitter and a pedometer. In addition, it has a companion app that is installed in the Smartphone and is called Fundo Bracalet [7]. The communication between booth is done by the Bluetooth LE protocol. By using the Smartwatch together with this companion app, it acquires the following features:

- Automatically update the date of the Smartwatch to the date of the Smartphone.
- Automatically display the temperature, altitude, and UV of your current location in the Smartwatch. All of these values are transmitted from the Smartwatch to the Smartphone.
- Automatically display a notification in the Smartwatch when a new message or call arrives at the Smartphone.
- To beep the Smartwatch to localize it.
- To manage the alarm in the Smartwatch.
- Automatically display the pedometer information in the Smartphone.
- To take a picture controlling the shoot from the Smartwatch.

IV. Methodology

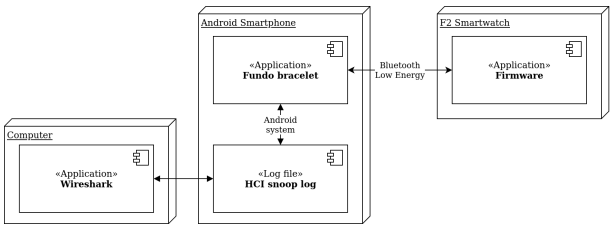

Fig. 1. Discovery stage deploy architecture

The methodology followed to develop the different components presented in this work, were split in two stages. In the first stage, that can be called the discovery stage, the packages was being discovered and the Python framework was developed side by side to test them. In the second stage, which began when all types of packets and the entire communication protocol were discovered, the mobile application was developed.

In order to discover the packages, was used and Android feature called Bluetooth Host Controller Interface (HCI) snoop log. This feature allow to captures all Bluetooth HCI packets in a file format that can be opened with Wireshark [9]. A

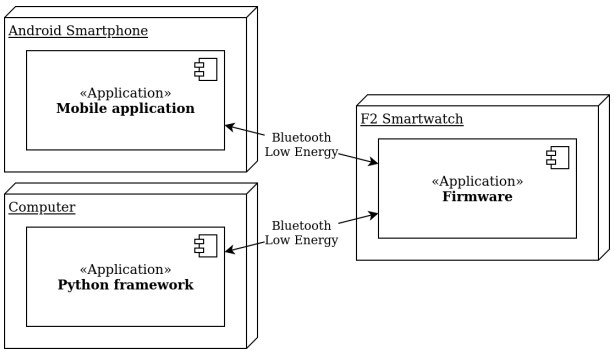

Fig. 2. Developed system deploy architecture

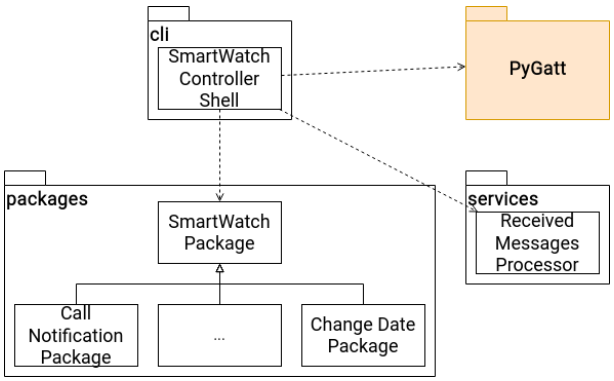

Fig. 3. Python framework architecture

diagram of the discovery stage deploy can be found in Figure 1.

Many of the features offered by the smartwatch can be activated by the user (e.g. changing the time). To discover the communication protocol used in these features, the action that activates the feature was repeated multiple times, and the generated Bluetooth logs were compared. For example, to discover the protocol used to change the time, a log was first obtained without changing the time. Then, two more logs were obtained, in the first, the time was set five times at 8:00 AM, in the second it was set ten times at this same hour. Then the records were compared, paying attention to whether there were any packages that did not appear in the first record, that appeared five times in the second record, and that appeared ten times in the third record.

Once a group of candidate packages was discovered, the test was repeated but trying to discover the contents of the package. To achieve this, several logs were generated by changing each of the fields that the package carried (hours, minutes, and seconds).

Other features, can not be triggered by the user (e.g. change the temperature). To discover these features, a periodic capture of logs was done, recording the value of the feature that wants to be discovered.

Once the entire communication protocol was discovered, the development of the android application began. To develop it, the Python Framework architecture was taken as a reference. Thus, the development of some components was limited to doing a translation between languages..

A diagram of the final system deploy can be found in Figure 2.

V. Framework to communicate with the smartwatch from the PC

The framework that allows the communication between the smartwatch and the PC, has the architecture shown in the Figure 3. On the one hand, it has a controller for the command line interface (CLI), which is in charge of solving the commands and handling the communication with the smartwatch. This communication is done using the PyGatt library.

The protocol used during communication has a 20 byte of maximum transmission unit (MTU). In the case of messages sent to the smartwatch, this is not a problem since all the packets are smaller. But some of the packages received as notifications had a bigger size than 20 bytes. To transmit these packets, the smartwatch fragments them before sending them, so they have to be reassembled in the framework before they can be interpreted. For this purpose, the ReceiveMessagesProcessor was created. Once a notification is received in the CLI controller, its content is transmitted to the ReceiveMessagesProcessor using a channel (in Python it is done using a Queue) where it is listened to from the framework. The ReceiveMessagesProcessor class is in charge of reassembling the packet and interpreting its content. The framework could have been designed to return the assembled package to the CLI controller, and make the interpretation of the package in it. But since only two types of packages are received as notifications, the proposed implementation was chosen in the name of simplicity.

VI. Replacement application for Android OS

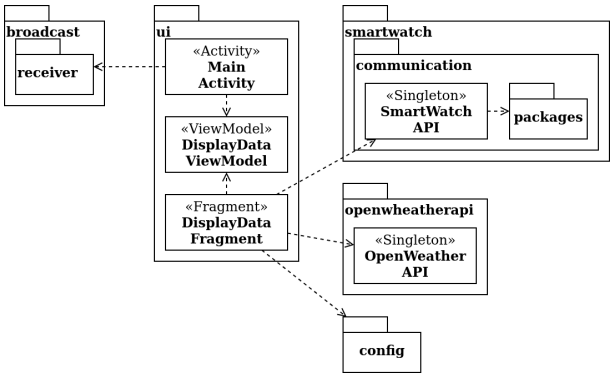

Fig. 4. Android application architecture

The application developed to replace the original, has the architecture shown in the Figure 3.

The mobile application is developed in the Kotlin programming language because it allows faster application development [12]. Also, it is composed of a single activity and makes

use of the Navigation Component [13] to display several screens if necessary (although the design raised makes use of one). The use of a single activity, allow to do there all the interaction with the Operating System services, and all notifications that it would provide are controlled through this Activity (new call notification, new message notification, and time change notification). To communicate this information from the Activity to the rest of the components of the application an Observer pattern is implemented. To obtain the weather information, the OpenWheatherAPI service is used. The openweatherapi package abstracts the network communications and provides a Kotlin interface. The smartwatch.communication package provides an abstraction layer over the communication with the smartwatch.

VII. Tests

Manual tests of the system were often carried out to ensure the quality of its different components. In the discovery stage, the system was continuously tested as part of the experimentation activity. These tests were also used to validate the Python framework. In the case of the Android application, the functionalities were tested as they were implemented, as well as a series of more exhaustive tests to search bugs when it was complete.

VIII. Conclusions

First of all, it can be concluded that reverse engineering this device was a complex task, and its success was based on the choice of the proper tests. Also, the process is mainly manual, and some features cannot be easily activated, such as updating the environmental parameters.

Secondly, the knowledge of the protocol has evidenced some of the design decisions of the original application, like the lack of a security layer in the communication (it may be an energy saving decision), or some bugs that have the firmware (e.g. the temperature can't admit negative numbers).

Finally, the mobile application developed not only replaces the original application but also opens up new possibilities for the smartwatch. One of these is the replacement of some of the functionalities it has by another of greater interest (e.g. the CO2 density could be shown instead of the altitude).

References

[1] Packages description.
https://github.com/AbelChT/Reverse-Engineering-No.1-F2/tree/master/protocol_documentation
[2] Python tool,
https://github.com/AbelChT/Reverse-Engineering-No.1-F2/tree/master/python_framework
[3] Mobile application for the Android OS,
https://github.com/AbelChT/Reverse-Engineering-No.1-F2/tree/master/android_application
[4] My journey towards Reverse Engineering a Smart Band — Bluetooth-LE RE,
https://medium.com/@arunmag/my-journey-towards-reverse-engineering-a-smart-band-bluetooth-le-re-d1dea00e4de2
[5] EU Directive 2009/24/EC,
https://eur-lex.europa.eu/legal-content/EN/ALL/?uri=CELEX%3A32009L0024
[6] Case C-406/10,
http://curia.europa.eu/juris/liste.jsf?num=C-406/10
[7] Fundo Bracelet Official APP,
https://play.google.com/store/apps/details?id=com.szkct.fundobracelet
[8] DT NO.I Official - Smartwatch Manufacturer Provide,
https://www.dtone.cc/
[9] Wireshark,
https://www.wireshark.org/
[10] Gattacker,
https://github.com/securing/gattacker
[11] Pygatt,
https://github.com/peplin/pygatt
[12] Kotlin in Android,
https://developer.android.com/kotlin
[13] Navigation Component Android,
https://developer.android.com/guide/navigation