The Independent IT-Security Institute
Magdeburg Germany

Internet of Things
Security Evaluation of nine Fitness Trackers

Dipl.-Inform. Eric Clausing
Michael Schiefer M.Sc.
Ulf Lösche
Dipl.-Ing. Maik Morgenstern

June 11, 2015
1.02

1 Introduction

Fitness trackers are becoming increasingly popular these days. Therefore it is hardly surprising there is a great number of different companies from different domains providing different products on a huge scale. Furthermore it is already possible for consumers in the US to get certain bonuses, benefits and rewards by wearing a fitness tracker and achieving predefined goals. These benefits can be for example discounts on health insurances, grants for therapies or vouchers for certain medical or health related products. Similar policies are already considered for certain other countries all over the world. As the trackers handle potentially sensitive data, they must be secured from data manipulation and theft – either by a malicious attacker or the owner himself. In our preliminary tests, we analyse the overall security concept of nine different trackers, in preparation of a detailed analysis. Our focus thereby is set on central questions concerning the security of data storage and transmission, either via Bluetooth or the internet, authentication on application and tracker and overall security flaws.

1.1 Motivation and similar work

Although the data collected and logged by the average fitness tracker – in most cases something like steps done, calories burned and hours slept – does not seem to represent sensitive data byword, there are still a whole lot of theoretical scenarios, where manipulation and/or data theft might lead to more or less serious threats to user privacy and data authenticity. An adequate security concept, for short range Bluetooth and internet communication, is therefore essential by all means.
With [Unucheck, 2015], [Barcena et al., 2014] and [Margaritelli, 2015] there is already work regarding security analysis of fitness trackers. In [Margaritelli, 2015], a detailed description of the Bluetooth security concept, and especially the authentication process, and its implementation for the Nike+ FuelBand is given. It is shown that a theoretically robust protocol can be easily ruined by poorly implementing it. Our tests show that this specific tracker and its security flaws is the rule rather than the exception.
In [Barcena et al., 2014], besides description of fitness tracking possibilities and security issues, apps working with smartphone only as well as apps working with tracker and the tracker itself are analysed. For example a Raspberry Pi is used as Bluetooth scanner to analyse the data packages sent and received by the tracker, with the result that most of these can be used to track the user. Additionally the network traffic is analysed too – according to [Barcena et al., 2014], 20 percent of the apps transmit passwords in plain text, but it is not mentioned which or even how many devices and apps are analysed.
In [Unucheck, 2015] there is another more general approach to the analysis by presenting the possibility of easily discovering and connecting different devices. There are also some obvious threats described, which are directly consequent to a missing security concept for the tracker. As an example [Unucheck, 2015] mentions

the theoretical possibility of exploiting heart rate data from a fitness tracker to observe consumer reactions on certain products or to analyse the influence of advertisement. Additionally to the possible threats mentioned in [Unucheck, 2015], there are a lot more scenarios in which unsecured trackers can lead to serious problems. Regarding user privacy for example, one can imagine a scenario where potential attackers might be able to surveil a persons movement and activity by observing their fitness data. With the theoretical Bluetooth communication range of about 100 metres, it is theoretically possible to easily scan a single-family house for persons present just by looking for active trackers. Along with the possibility of acquiring the fitness data from unsecured trackers it would be easy to determine whether the persons are active or asleep and along with the user information some trackers store, even what gender, age, height and weight these persons are. At this point at the latest, the threat becomes obvious.

But also regarding data authenticity and integrity there are some scenarios, which could lead to serious financial or health related damage. As mentioned above, there are already countries in which fitness trackers are used or are to be used to reward activity with e.g. discount on health insurance contributions. Using trackers without a solid security concept, or as in [Margaritelli, 2015] described, with a poor implementation of said concept for such purposes could be highly disadvantageous for insurer and/or insuree. For example a dishonest but competent insuree would theoretically be able of achieving all requirements for being rewarded just by digitally altering the data stored on the tracker. And that is without even lifting a finger. The poor security concept of some trackers even allow for the sharing of activity data between different accounts. In this case the damage caused by data forgery could be enormous.

The other way around, the days or even months work of an honest and active insuree could be destroyed within seconds by resetting all data on his tracker – extremely annoying at the least and a considerable financial loss most certainly.

Additionally to what was already mentioned, there are a whole lot of other possibilities of manipulating the trackers function by setting or resetting alarm timers, trigger vibration, alter system time and so on. Although non of these actions will directly cause a serious damage, they are disturbing and annoying nevertheless.

2 Test Overview

This chapter describes the test as well as the used devices and apps. The presented tests are hands on – they are meant as a first overview and preparation for further investigation of the tracker and app implementation.

2.1 Test Setup

As a result of the simple nature of a hands-on test, the test setup is also simple, see figure 2.1. The apps run on android 4.4.4 or 5.0.1. The smartphone communicates via wireless local area network and its communication is captured. The trackers use Bluetooth or Bluetooth Smart to communicate with the smartphone and app respectively. The original apps as well as a self-made app is used for testing.

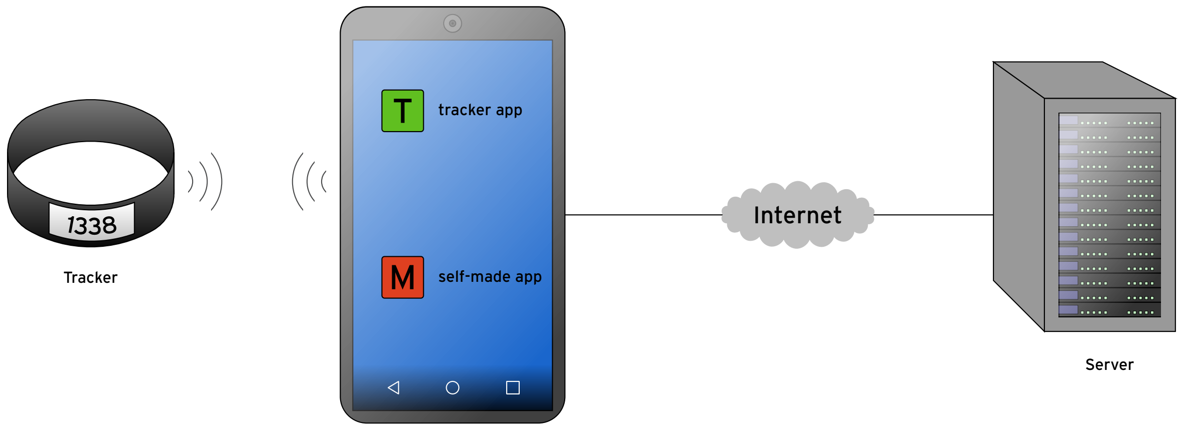

Figure 2.1: Schematic test setup

2.2 Test procedure

Our tests consider the Bluetooth connection as well as the internet connection and the app itself. We capture all data sent to or received from the internet and search for unencrypted information – for this first test, we do not attempt to break the encrypted data streams. The unencrypted data however can be read and potentially altered from every node between sender and receiver. As fitness data can be considered as potentially sensitive data, it should remain private and therefore should only be transmitted in an encrypted way.

Bluetooth is a wireless communication mechanism which can be initiated by every Bluetooth compatible device nearby. In many cases this can be done without the tracker owner noticing. If the tracker then does not implement any additional security features to protect its content, it can be read and/or manipulated.

We test each tracker for its security measures, Bluetooth visibility as well as its connection characteristics to determine its basic capability to resist potential attacks. This includes an analysis of the pairing process, i.e. if you need to press a button or read a pin from the tracker to pair it with your smartphone or if a connection can be established without confirmation of the trackers owner. Additionally we check for a mutual authentication between tracker and smartphone as it ensures that none of both is communicating with a malicious device.

Furthermore the apps themselves are analysed. Depending on the gathered information, a self-made app tries to access the tracker, at least for some of the products. Because of the potential time intensive character of a detailed analysis, only some devices were tested with the self-made app.

2.3 Tested products

For our tests, we choose fitness trackers with integrated pedometers. Most of them are popular and widely used or the producer is well known. As we only consider devices available in Germany, with apps running on android phones of different manufactures, fitness trackers like the Microsoft Band and the Samsung tracker are excluded from our tests. Table 2.1 shows the chosen devices and their features.

	Acer Liquid Leap	FitBit Charge	Garmin Vivosmart	Huawei TalkBand B1	Jawbone UP24	LG Lifeband Touch FB84	Polar Loop	Sony Smartband Talk SWR30	Withings Pulse O_x
Bluetooth Classic	✗	✓	✓	✓	✗	✓	✗	✓	✓
Bluetooth Smart/Low Energy	✓	✓	✓	✗	✓	✓	✓	✗	✓
NFC	✗	✓	✗	✓	✗	✗	✗	✓	✗
Pedometer	✓	✓	✓	✓	✓	✓	✓	✓	✓
Sleep	✓	✓	✓	✓	✓	✗	✓	✓	✓
Display	✓	✓	✓	✓	✗	✓	✓	✓	✓
Heart rate/pulse	✗	✗	✗	✗	✗	✗	✗	✗	✓
Oxygen	✗	✗	✗	✗	✗	✗	✗	✗	✓
GPS	✗	✗	✗	✗	✗	✗	✗	✗	✗

✓Available ✗Not available

Table 2.1: Overview about the features of the different trackers and their corresponding apps

As already mentioned every device integrates a pedometer. Different sources in the internet proclaim that Withings Pulse O_x communicates with Bluetooth Smart, but according to our tests it seems to use Bluetooth Classic even though it is capable of Bluetooth Smart. Only the devices of FitBit, Huawei and Sony support NFC but mostly for starting the app and not for data transfer itself. Only the LG Lifeband seems not to support sleep tracking according to the missing information on the vendor site and options provided by the app. The Jawbone UP24 is the only device without an display – only two LEDs help to identify the status of the band. Some of the other devices support touch or use an electronic ink display, like Acer or Sony. Of the tested devices only Withings is capabale of measuring the heart pulse and oxygen saturation of the blood. None of the analysed systems use GPS.

Table 2.2 lists the name of the used apps as well as their versions grouped by the corresponding tracker. Sony is the only manufacturer in the test which makes it necessary to install two apps, one for the communication to the tracker and one for actually using the data received from the tracker. During our analysis, new versions of the specific apps happened to be released. In some cases we installed the newer versions as well and checked for changes. Accordingly we list both versions of the specific app in 2.2.

App name	Version
Acer Liquid Leap	
Leap Manager	1.0.292p
FitBit Charge	
Fitbit	2.4.2
Garmin Vivosmart	
Garmin Connect Mobile	2.11.2
Huawei TalkBand B1	
Huawei Wear	12.03.02.01.00
Huawei Wear	12.04.03.01.00
Jawbone UP24	
UP - Requires UP/UP24/UP MOVE	4.2.0
LG Lifeband Touch FB84	
LG Fitness	2.5.23
Polar Loop	
Polar Flow	2.1.0
Sony Smartband Talk SWR30	
Lifelog	2.6.A.0.10
SmartBand Talk SWR30	3.0.0.102
Withings Pulse O_x	
Health Mate	2.04.40
Health Mate	2.04.50

Table 2.2: List of used app and version depending on the tracker

3 Analysis

The results of the hands-on tests are presented in table 3.1. In the following the different observations are explained and trackers and/or apps, which attract attention in one way or another, are referred to. All trackers and apps are examined in a state after complete initialisation and setup.

	Acer Liquid Leap	FitBit Charge	Garmin Vivosmart	Huawei TalkBand B1	Jawbone UP24	LG Lifeband Touch FB84	Polar Loop	Sony Smartband Talk SWR30	Withings Pulse O_x
Bluetooth									
deactivatable at tracker	✗	✗	✓	✗	✗	✓	✗	✗	✗
pairing requires hardware access	✗	✗	✗	✗	✓	✓	✓	✓	✓
app needs to be authenticated	✗	✗	✓	✓	✓	✓	✓	✓	✓
tracker only usable with one smartphone	✗	✗	✗	✗	✗	✗	✓	✓	✗
tracker is always invisible	✗	✗	✗	✓	✗	✗	✓	✓	✓
tracker visibility is hardware activated	✗	✗	✗	✓	✓	✗	✓	✓	✓
Miscellaneous									
obfuscation	✗	≈	✗	≈	✓	✗	✓	✓	≈
absence of log entries	✗	✗	✓	✗	✓	✗	✗	✓	✗
app is published as full release	✗	✓	✓	✗	✓	✗	✗	✓	✗
data is privately saved on smartphone	✓	✓	✓	✓	✓	✓	✓	✓	✓
internet communication is encrypted	≈	✓	✓	≈	✓	✓	✓	✓	✓

✓High ≈Medium ✗Low/Not existent

Table 3.1: Overview of preliminary analysis results for trackers and their corresponding apps

3.1 Bluetooth

In this section, we review and explain all Bluetooth related security features and their implementation on tracker and application side. Table 3.1 lists the features, we considered especially relevant because a poor implementation of these on either tracker or application side can result in a leakage of private data and/or manipulation of the trackers function.

Bluetooth deactivation

The first point on the list is the easiest and most intuitive way to protect the data on your tracker against a malicious attack and also the easiest to implement – the deactivation of the Bluetooth hardware on tracker side. Without the possibility to communicate with the tracker, an attacker has no chance of retrieving or manipulating any data located on that tracker. Although this option seems to be an intuitive and simple way of protecting your tracker against an unauthorized access, only two of the tested devices – Garmin Vivosmart and LG Lifeband Touch – implement such an option, simply be allowing the user to deactivate Bluetooth through the tracker menu. The Huawei TalkBand B1 technically supports the deactivation of Bluetooth but it is only triggered if it cannot reach the paired smartphone for more than three minutes. It is not possible to manually deactivate Bluetooth.

Pairing

According to Bluetooth communication protocol a connection is initiated by firstly pairing the devices, i.e. introducing the two communication partners to each other. After pairing, the connection can be established and information can be transferred. Accepting a pairing request corresponds to accepting the connection to the corresponding device. As the requesting device might possibly be an attacker or at least an unauthorized entity, it may be wise not to accept any pairing requests and let the trackers owner explicitly decide whether the pairing should be accepted or not. The second point on the list describes this option of manually accepting or rejecting a pairing request and thereby eliminating the possibility of a connection to the tracker without the user even knowing. Five of the tested devices ensure that the pairing process is intended by the legitimate user by a number of different measures. These measures can be for instance a confirmation by simply pressing 'OK', in case of LG Lifeband Touch, inserting a number displayed on the device or a pairing via NFC as Sony's Smartband SWR30 does. The Acer device shows a number on its display which should serve as a pairing pin but as this pin is only 4 digits from the devices public name, the pairing procedure cannot be considered as robust. In general the specific concept of a secure pairing process is open, it just has to ensure that the tracker hardware and the connecting device are both in the hands of the legitimate user. In case of the Acer device this cannot be ensured.

Tracker authentication

In digital communication it is always important to be able to determine the identity of the connection partner in question. The reviewed process here is no exception from the rule. As it might be possible for an attacker to disguise himself as a fitness tracker and thereby getting a connection to a smartphone via the tracker application, it is important to provide an adequate method which allows for an authentication of the fitness tracker before actually communicating. In our tests at least six of the trackers use such an authentication process in one way or another. For the Huawei, Sony and Acer devices we cannot find enough clues to make a conclusion, while it is very likely for the Huawei and Sony devices to implement one. Without further research, we are forced to speculate at this point and therefore exclude this aspect from our test results.

Smartphone or Application authentication

The other way around, it is also important to be secured against an illegitimate connection request from a smartphone respectively app which is not authorized to communicate with the tracker. For this case, the tracker must be able to determine the identity of the connection initiator. At least four of the tested trackers have the ability of authenticating the app they are communicating with. The Jawbone UP24 for example does a challenge-response authentication. It sends a byte sequence to the app which then has to use a secret byte array to calculate an MD5 byte sequence which is then sent back to the tracker. The concept itself seems to be robust, whether or not this authentication procedure works as intended depends on the actual implementation. In contrast, we have the Fitbit Charge, which does not use any authentication on tracker side at all and carelessly provides the saved fitness data to everyone asking for it. The code snippet in listing 3.1 shows how easy it is for an potential attacker to get the latest fitness data from the Fitbit tracker without any form of authentication or knowledge of the app sources. Figure 3.1 shows an exemplary packet as it is received from the tracker and in comparison the corresponding representation by the official app. Once connected to the device the attacker is even notified whenever the fitness data changes and thus can even determine whether the target is currently moving, standing still (indicated by changing step count) or going up or down a stairway (indicated by changing floor count). Regarding the theoretical range for Bluetooth communication of about 100 metres, this is a real problem. Imagine the same tracker with GPS functionality and the whole thing gets even more disturbing.
At a first view it seems that the Acer app needs to be authenticated – before the tracker provides access to its data, some packages need to be transferred and received. But actually there are some cases in which the tracker sends the step count data before the connection to the device is even confirmed. The Jawbone and Huawei device as another example allow for a sharing of the step counts between different accounts. You can simply install the same tracker on different devices and the tracker synchronises its data with every app even if different user accounts are used. However the device of Withings will only synchronise its data with the last paired smartphone, you can use it again with a previously paired device and set up the app. Then the app will automatically find the device and reinitialize it without questioning, leading to a reset of the device and loss of all tracked data.

```
// ... Initialize Bluetooth LE scanning via standard Bluetooth LE protocol
// ... Establish connection to "Charge" via standard Bluetooth LE protocol
// ... Discover services running on tracker via standard Bluetooth LE protocol

public void onServicesDiscovered(BluetoothGatt gatt, int status) {
        //Fitness data service; UUID from service discovery
        BluetoothGattService service = gatt.getService(UUID.fromString("558dfa00-4fa8-4105-9f02-4eaa93e62980");

        //Enable notifications to retrieve fitness data whenever it has changed;
        BluetoothGattCharacteristic serviceCharacteristic = service.getCharacteristic(UUID.fromString("558dfa01-4fa8↩
          -4105-9f02-4eaa93e62980"));

        setCharacteristicNotification(gatt, serviceCharacteristic, true);
        // ... Be notified whenever updated fitness data is available
}

public void onCharacteristicChanged(BluetoothGatt gatt, BluetoothGattCharacteristic characteristic) {
        //Fetch the data
        byte[] data = characteristic.getValue();
}
```

Listing 3.1: Code snippet for retrieving fitness data from Fitbit Charge

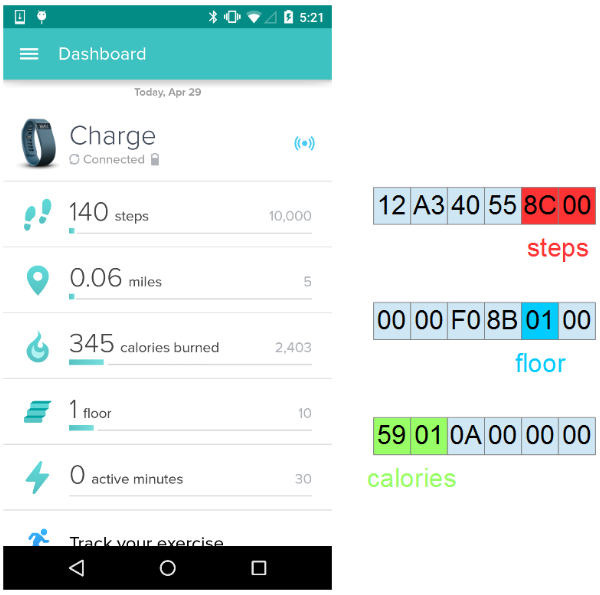

Figure 3.1: Byte order of a Fitbit Bluetooth packet as it can directly be received from the tracker; in comparison the representation of the same data in app

Tracker visibility

The best way to avoid unauthorized access to the tracker, is to disguise its presence. This can be done by completely deactivating the Bluetooth hardware or by ensuring the tracker cannot be found despite Bluetooth being activated. In our tests there are three trackers, namely the Sony, Polar and Withings devices, which remain practically invisible for Bluetooth scanning once they are set up and bonded to a smartphone.

The Sony Smartband for example only reconnects to the known bonded device via NFC and is invisible to pretty much everything else. The Polar Loop is invisible until it is explicitly set to 'active' to allow for a synchronisation. But even set to active, it does not accept connections from random devices. The Withings tracker only synchronizes autonomously to the smartphone it was last bonded. You can see that device, only if it tries to synchronize. The Huawei TalkBand B1 deactivates its Bluetooth after it loses its connection to the bonded smartphone for more than three minutes. Afterwards a press on the button of the TalkBand is necessary to reactivate Bluetooth. The Jawbone band is invisible for some time and you have to press the button or just move the band to set it visible. But after that it is visible for quite a time – for more than an hour.

3.2 Miscellaneous

In this section we review the aspects of app implementation, data storage and information propagation which can result in data leakage or a simplified reverse engineering of the application and communication.

Code obfuscation

The easiest but nonetheless a quite effective way for developers to hinder the reverse engineering of their product, is code obfuscation. Implemented and correctly applied, obfuscation can be enough to discourage a low skill attacker from even trying. But also skilled attackers might have at least a harder time discovering all detailed aspects of the focused application. So before releasing an application which handles potentially sensitive data, it should be obfuscated no matter what. In our tests six out of nine applications used code obfuscation in a more or less consequent way. The three other apps not using code obfuscation – from Acer, Garmin and LG – had their communication protocol outsourced to shared libraries, which also can be an effective way to prevent code analysis. Although it might be possible to use these libraries in an attackers app without the necessity to know how they work. From all reviewed apps, only the Polar and Sony Lifelog app use both code obfuscation and a shared library with outsourced communication. The quality of the obfuscation differs between apps but is in most cases sufficiently good. Of all analysed apps the Jawbone UP24 has the highest level of obfuscation, far above most of the other apps. But although the obfuscation for the Jawbone is very high, there is one big flaw, which seems to went unseen through final release checking – some of the original class names are still included, plain text and unmistakably relatable to the corresponding classes.

Logging

Missing code obfuscation can make reverse engineering pretty easy but there is a point which can help even more – excessive logging and remaining debug statements in releases. For the development process a detailed logging is essential to be able to track down bugs, supervise running processes and check interim results, but in a release version they make reverse engineering easy. As the results from table 3.1 show, in our tests we only have a few apps which completely strip away all debug statements and logging for the release. In the case of Polar and LG one is almost capable of reconstructing the whole communication process without even taking a look at the source code. The logging of these two apps include method calls, sent and received data and a lot of other useful information for an app analysis. Listing 3.2 shows the standard logcat output of the Polar Flow app, when synchronizing with the tracker. The logging of the LG Fitness app looks quite similar regarding given detail.

```
04-16 04:29:16.517: I/BluetoothClassifier(2565): Bluetooth Device Name: Polar Loop 6B42AF1B
04-16 04:29:16.549: D/BluetoothService(5064): BluetoothService.pauseScan
04-16 04:29:16.549: D/BluetoothService(5064): pauseScan()
04-16 04:29:16.549: D/BluetoothAdapter(5064): stopLeScan()
04-16 04:29:16.550: D/BluetoothAdapter(5064): scan not started yet
04-16 04:29:17.269: D/BtGatt.btif(1875): btapp_gatts_handle_cback: Event 2
04-16 04:29:17.269: D/BtGatt.GattService(1875): onAttributeWrite() UUID=00002902-0000-1000-8000-00805f9b34fb, serverIf=6, ↩
  type=3
04-16 04:29:17.270: D/BluetoothService(5064): -> Writing to CCC: 0100
04-16 04:29:17.271: D/BtGatt.btif(1875): btif_gatts_send_response
04-16 04:29:17.271: D/BtGatt.btif(1875): btgatts_handle_event: Event 2012
04-16 04:29:17.271: D/BtGatt.GattService(1875): onResponseSendCompleted() handle=43
04-16 04:29:21.328: D/btif_config_util(1875): btif_config_save_file(L188): in file name:/data/misc/bluedroid/bt_config.new
04-16 04:29:24.308: D/b(5064): onReceive intent.getAction() = com.polar.pftp.DEVICE_READY_FOR_PFTP
04-16 04:29:24.308: D/b(5064): ACTION_DEVICE_READY_FOR_PFTP
04-16 04:29:24.309: D/SyncService(5064): SyncService.onStartCommand: START_SYNC
04-16 04:29:24.310: D/SyncService(5064): Current running PrimarySyncTasks:
04-16 04:29:24.310: D/SyncService(5064): -> none
04-16 04:29:24.310: D/SyncService(5064): Current running SyncTasks:
04-16 04:29:24.310: D/SyncService(5064): -> empty
04-16 04:29:24.311: D/SyncService(5064): Executing PrimarySyncTask
04-16 04:29:24.313: D/SST(5064): Running primary task ActionThread-START_SYNC=fi.polar.polarflow.service.sync.q@3290f54b
04-16 04:29:24.313: D/SyncService(5064): Starting sync
04-16 04:29:24.313: D/DeviceSync(5064): Device sync started
04-16 04:29:24.314: D/PFTP(5064): PFTPController.sendSyncStartNotification()
04-16 04:29:24.316: D/PFTP(5064): Write 4 bytes: 0A00 0000
04-16 04:29:24.317: D/BtGatt.btif(1875): btif_gatts_send_indication
04-16 04:29:24.317: D/BtGatt.btif(1875): btgatts_handle_event: Event 2011
04-16 04:29:24.317: D/BtGatt.btif(1875): btapp_gatts_handle_cback: Event 5
04-16 04:29:24.317: D/BtGatt.GattService(1875): onNotificationSent() connId=262, status=0
04-16 04:29:24.518: D/PFTP(5064): KeepConnectionAlive turned on
04-16 04:29:24.518: D/BluetoothService(5064): BluetoothService.sendSyncStart
04-16 04:29:24.522: D/SyncTimes(5064): SyncInfoProto$SyncInfoSyncTask started.
04-16 04:29:24.524: D/PFTP(5064): PFTPController.loadData(/SYNCINFO.BPB)
04-16 04:29:24.535: D/PFTP(5064): Write 20 bytes: 1900 1100 0800 120D 2F53 594E 4349 4E46 4F2E 4250
04-16 04:29:24.536: D/BtGatt.btif(1875): btif_gatts_send_indication
04-16 04:29:24.536: D/BtGatt.btif(1875): btgatts_handle_event: Event 2011
04-16 04:29:24.536: D/BtGatt.btif(1875): btapp_gatts_handle_cback: Event 5
04-16 04:29:24.537: D/BtGatt.GattService(1875): onNotificationSent() connId=262, status=0
04-16 04:29:24.538: D/PFTP(5064): Write 3 bytes: 0042 00
04-16 04:29:24.539: D/BtGatt.btif(1875): btif_gatts_send_indication
04-16 04:29:24.539: D/BtGatt.btif(1875): btgatts_handle_event: Event 2011
04-16 04:29:24.539: D/BtGatt.btif(1875): btapp_gatts_handle_cback: Event 5
04-16 04:29:24.539: D/BtGatt.GattService(1875): onNotificationSent() connId=262, status=0
04-16 04:29:24.604: D/BtGatt.btif(1875): btapp_gatts_handle_cback: Event 2
04-16 04:29:24.604: D/BtGatt.GattService(1875): onAttributeWrite() UUID=fb005c16-02e7-f387-1cad-8acd2d8df0c8, serverIf=6, ↩
  type=2
04-16 04:29:24.605: D/PFTP(5064): Read 20 bytes from buffer: 1900 0000 0A13 0A07 08DF 0F10 0418 1012 0608 0810
04-16 04:29:24.605: D/BtGatt.btif(1875): btapp_gatts_handle_cback: Event 2
04-16 04:29:24.605: D/BtGatt.GattService(1875): onAttributeWrite() UUID=fb005c16-02e7-f387-1cad-8acd2d8df0c8, serverIf=6, ↩
  type=2
04-16 04:29:24.606: D/PFTP(5064): Read 9 bytes from buffer: 001D 1811 1801 2000 00
04-16 04:29:24.607: D/fi.polar.polarflow.data.ProtoEntity(5064): getProtoBytes
```

```
04-16 04:29:24.607: D/fi.polar.polarflow.data.ProtoEntity(5064): getProtoBytes return cached: 23
04-16 04:29:24.616: D/fi.polar.polarflow.data.ProtoEntity(5064): getProtoBytes
04-16 04:29:24.618: D/SQL Log(5064): SQLiteQuery: SELECT * FROM TRAINING_COMPUTER WHERE SYNC_INFO_PROTO = ?
04-16 04:29:24.620: D/SQL Log(5064): SQLiteQuery: SELECT * FROM DEVICE_SENSOR_LIST WHERE id=? LIMIT 1
04-16 04:29:24.622: D/SQL Log(5064): SQLiteQuery: SELECT * FROM DEVICE_INFO_PROTO WHERE id=? LIMIT 1
04-16 04:29:24.623: D/SQL Log(5064): SQLiteQuery: SELECT * FROM TRAINING_COMPUTER_LIST WHERE id=? LIMIT 1
04-16 04:29:24.624: D/SQL Log(5064): SQLiteQuery: SELECT * FROM USER_DEVICE_SETTINGS WHERE id=? LIMIT 1
04-16 04:29:24.626: D/SQL Log(5064): SQLiteQuery: SELECT * FROM SYNC_INFO_PROTO WHERE id=? LIMIT 1
04-16 04:29:24.628: D/SQL Log(5064): SQLiteQuery: SELECT * FROM USER WHERE REMOTE_IDENTIFIER = ?
04-16 04:29:24.631: D/SQL Log(5064): SQLiteQuery: SELECT * FROM USER_PREFERENCES WHERE id=? LIMIT 1
04-16 04:29:24.632: D/SQL Log(5064): SQLiteQuery: SELECT * FROM USERUSER_ID WHERE id=? LIMIT 1
04-16 04:29:24.634: D/SQL Log(5064): SQLiteQuery: SELECT * FROM ORTHOSTATIC_TEST_LIST WHERE id=? LIMIT 1
04-16 04:29:24.635: D/SQL Log(5064): SQLiteQuery: SELECT * FROM RECOVERY_TIMES_PROTO WHERE id=? LIMIT 1
04-16 04:29:24.637: D/SQL Log(5064): SQLiteQuery: SELECT * FROM DAILY_ACTIVITY_SAMPLES_LIST WHERE id=? LIMIT 1
04-16 04:29:24.638: D/SQL Log(5064): SQLiteQuery: SELECT * FROM DEVICE_SPORT_LIST WHERE id=? LIMIT 1
04-16 04:29:24.640: D/SQL Log(5064): SQLiteQuery: SELECT * FROM USER_PHYSICAL_INFORMATION WHERE id=? LIMIT 1
04-16 04:29:24.642: D/SQL Log(5064): SQLiteQuery: SELECT * FROM FITNESS_TEST_LIST WHERE id=? LIMIT 1
04-16 04:29:24.643: D/SQL Log(5064): SQLiteQuery: SELECT * FROM DAILY_ACTIVITY_GOAL WHERE id=? LIMIT 1
04-16 04:29:24.645: D/SQL Log(5064): SQLiteQuery: SELECT * FROM TRAINING_SESSION_TARGET_LIST WHERE id=? LIMIT 1
04-16 04:29:24.646: D/SQL Log(5064): SQLiteQuery: SELECT * FROM SPORT_PROFILE_LIST WHERE id=? LIMIT 1
04-16 04:29:24.648: D/SQL Log(5064): SQLiteQuery: SELECT * FROM FAVOURITE_TRAINING_SESSION_TARGET_LIST WHERE id=? LIMIT 1
04-16 04:29:24.649: D/SQL Log(5064): SQLiteQuery: SELECT * FROM JUMP_TEST_LIST WHERE id=? LIMIT 1
04-16 04:29:24.651: D/SQL Log(5064): SQLiteQuery: SELECT * FROM TRAINING_COMPUTER_LIST WHERE id=? LIMIT 1
04-16 04:29:24.652: D/SQL Log(5064): SQLiteQuery: SELECT * FROM RR_RECORDING_TEST_LIST WHERE id=? LIMIT 1
04-16 04:29:24.654: D/SQL Log(5064): SQLiteQuery: SELECT * FROM PLANNED_ROUTE_LIST WHERE id=? LIMIT 1
04-16 04:29:24.655: D/SQL Log(5064): SQLiteQuery: SELECT * FROM TRAINING_SESSION_LIST WHERE id=? LIMIT 1
04-16 04:29:24.674: I/Sugar(5064): User saved : 1
04-16 04:29:24.674: D/fi.polar.polarflow.data.ProtoEntity(5064): getProtoBytes
```

Listing 3.2: Logcat logging snippet of the Polar Flow app synchronizing with Loop tracker

Another noticeable thought about logging is, that it can revert at least parts of the obfuscation, e.g. if a function is obfuscated but starts with a logging call mentioning the name of the function. This is still a problem if the logging is deactivated but the logging remains in code. There are several apps in the tests where this is true for at least some functions.

Debug vs. Release

Published apps should not contain any debug information at all. While these information could help to find bugs in the code, they could also help to reverse engineer the app. None of the evaluated apps and versions has set the `android:debuggable` option in its manifest. But the app for Acer Liquid Leap ships with a debug and release version of the library handling the tracker communication and is loading the debug version. The Withings and Huawei devices do not contain debug libraries but produce a lot of log entries marked as debug.

Data storage

As the analysed applications handle private data, it is important to check whether or not the persistent storage on the phone is secured to prevent other apps from freely accessing the data. Normally the android system itself ensures that the access to application data is protected by an adequate rights management and data storage is only assigned to secured locations. Nevertheless applications can explicitly decide for themselves to use a public saving location on the phone. According to our findings, none of the analysed applications does so and all sensitive user data is exclusively stored in secured locations. Some apps like Garmin use the SD card and public saving locations for caching certain contents, but none of these are sensitive by any means.

Interesting for rooted phones is the fact that some of the apps save sensitive authentication data in plain text in the corresponding app data locations. As on rooted phones theoretically each app can access the app data of every other app, this might be a threat for some users. The Garmin Connect Mobile app for example stores information like secret user tokens, transaction keys and IDs in a plain text .xml-file in its app folder. In the Polar app folder, one can even find the complete user account information including username, ID and plain text password.

Online synchronisation

As the data collected, profiled and stored for each user is also synchronized with the corresponding app cloud over the internet, the security of the online communication must be reviewed as well. In our tests, we just check whether or not sensitive data like fitness and user data is transferred in an encrypted way, i.e. via HTTPS, or unencrypted way, i.e. via HTTP, to the corresponding synchronization server. In our findings none of the analysed apps completely foregoes encryption and nearly all of them send the sensitive data over secured channels, although it is quite safe to say that there is a lot of other data sent over HTTP, at least by some apps. For example the Withings app loads websites, e.g. with tipps, without encryption. The Leap Manager from Acer sends some of the tracked data in plain text through HTTP and receives some statistics unencrypted. The Huawei App seems to send all data via SSL but it searches for updates in plain text using complete URLs in the answer. It was not tested if this update mechanism could be compromised.

4 One exemplary product in detail

In this section we present a more detailed view and explain ways to manipulate one exemplarily chosen device – namely the Acer Liquid Leap. This device is specifically suited as an example, as it reveals the obvious problems resulting from a insufficiently designed security concept and poor implementation.

Pairing and visibility

Before the tracker is initiated its Bluetooth is only active if the device is activated by touching the display multiple times or if the tracker is charged. To perform a first initialisation of the device the app asks for something that looks like a pin of four hexadecimal digits. These digits are displayed by the tracker when charging and have to be inserted into the app so that it can find and initialize the device. But as our tests show, these digits are only part of the devices public name, namely the 13th and 14th as well as the 17th and 18th character. The actual problem is, that the device name itself can be retrieved by anyone with a Bluetooth phone using the standard Bluetooth device scan, see figure 4. Along with the knowledge of the obviously hard coded character positions, no further hardware access is necessary to pair with the tracker.

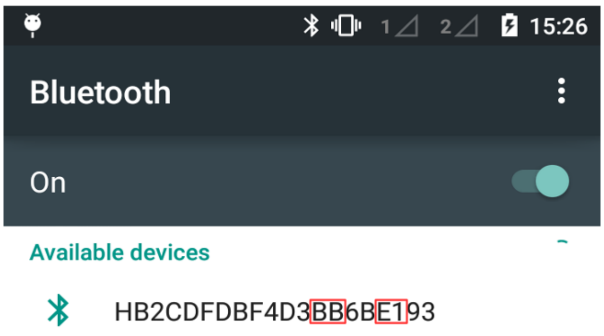

Figure 4.1: Device name received via standard Bluetooth scan; the characters in red boxes are used for the pairing process

Authentication, logging and debug state

Once the device is paired the app initializes the tracker by setting the user on it. Afterwards the tracker continuously responds to Bluetooth LE scans if it is not connected to a smartphone. The app itself does not really talk to the tracker, only the Bluetooth interface of android is used. The data seems to be transferred to a specific library, which interprets the data from the tracker and generates the packages sent to it. Thus everyone with the library, so obviously anyone with access to the play store, can communicate with the tracker and interpret its data without caring about a possibly existent authentication procedure performed by the library. Additionally the app is shipped with a release and a debug version of that library and is obviously using the debug version, resulting in a lot of debug output in the logcat of android. Listing 4.1 shows a short cutout of the logcat output from the original app. The first line shows the byte array of a received packet in decimal. It is 22 byte long and in a so called SDK format. The first two bytes represent the Bluetooth Gatt Characteristic from which the data was received and the last 20 bytes are the Bluetooth packet. In line 5 one can see the interpretation and output of the data performed by the original library – 342 walked steps or in byte split `[1] [86]`, zero stairs climbed up and 667 or `[2] [155]` steps run. It displays 480 seconds of activity, represented in minutes `[0] [8]` and 860 respectively `[3] [92]` calories burned. The format of the about 2.62 miles is unknown. The byte-split numbers can also be found in the raw packet printed in the first line.

```
1  04-15 09:13:07.718: D/PeripheralManager(5830): Java Received  [255]  [241]  [7] [0] [1] [86] [0] [2] [155] [0] [0] [0] [8]↩
     [2] [137] [36] [3] [92] [0] [0] [4] [1]
2  04-15 09:13:07.718: I/native-activity(5830): PEBBLE Process Packet
3  04-15 09:13:07.718: D/native-activity(5830): Received SDK log to write
4  04-15 09:13:07.718: I/native-activity(5830): SDK: [3437218] CALLBACK RECEIVED: 1 (activity)
5  04-15 09:13:07.728: D/native-activity(5830): JNI activitydata walking = 342, stairs = 0, running = 667, activitySeconds = ↩
    480, miles = 2.622790, calories is 860.000000
6  04-15 09:13:07.728: I/PEDO(5830): stats: 342,667,0,0,480.0,2.6227903366088867,860.0
7  04-15 09:13:07.728: V/BackgroundService(5830): processing challenges and milestones
8  04-15 09:13:07.738: V/BackgroundService(5830): sending activity data to activity
9  04-15 09:13:07.738: D/BackgroundService(5830): Updated Activity steps:1009 distance:2.6227903366088867 seconds:480 ↩
    calories:860.0
10 04-15 09:13:07.738: V/PeripheralManagerWrapper(5830): activitydata is Walking:342 Stairs:0 Running:667 Seconds:480 Miles↩
    :2.6227903366088867 Calories:860.0
11 04-15 09:13:07.738: V/TAG_Striiv_MainActivity(5830): Received activity data from background service
12 04-15 09:13:07.738: I/TAG_Striiv_MainActivity(5830): updating tiles with activity data
```

Listing 4.1: Some output from logcat from the original app showing tracker data

Manipulation

As our tests show, it is possible to modify various kinds of user data, without changing any source code but simply by using the original app, which obviously lacks any kind of plausibility check – we were able of setting the step length to several metres, so that walking 1000 steps lead to a walk length of multiple kilometres. Even the cloud server in the background does not seem to complain and saves these absurd information. The example in listing 4.1 visualizes the manipulation possibilities: All in all there are 1009 steps done, burning unrealistic 860 calories, covering a distance of about 2.62 miles or 4.22 kilometres, leading to more than four metres per step. It is most likely that many more manipulations are possible once the included library is utilized. Changing date and time, setting alarm timers and nearly every other aspect of the tracker might be manipulated in that way – resulting in heavy annoyance for the user at the least. But it is not only possible to alter data creation, it is also possible to remotely remove the existing user on the device leading to an uninitialized device. By synchronizing the device with the initialized app – which is a bit tricky because an uninitialized tracker is not as communicative as an initialized one – the device will be reinitialized automatically. But the tracker will start by zero again and at least all data of that day up to that point is lost. Unsynchronized data of the last days will be lost, too.

As one can see the Acer Leap suffers from a lot of vulnerabilities, which could have been avoided or at least mitigated with a little more effort in regards to an adequate security concept and final release checking. As it is implemented now, the tracker and the data located on it is basically unprotected from all kinds of malicious attacks and as the Acer Liquid Leap seems to be a rebranded 'striiv' device, these analysis results may probably apply for a lot more devices and apps out there.

5 Summary and upcoming investigation

In this analysis, we test nine fitness trackers on overall security and privacy conservation. We create a list of aspects to be tested and define points especially important in our opinion. Our main goal thereby is to find out, how easy it is to get the acquired data from the tracker either directly via Bluetooth, by eavesdropping the connection to the cloud when synchronisation is executed, or by reading out the internal storage. The points on our list define the requirements towards a tracker-app combination to protect from these three possibilities. Especially important thereby are the aspects of mutual authentication between tracker and app, the encryption of sensitive internet communication and secure storage of all relevant data. Regarding the authentication, we not yet rate the actual robustness of the used methods but only mark which trackers and apps explicitly dedicate to perform an authentication and which do not. The other points on the list are not that crucial in comparison to the three mentioned above but nevertheless can make a huge difference in combination.

As one can see in table 3.1, the results are quite mixed up. We discovered sophisticated security concepts, no security concepts at all and almost every possible alternative in between. The Sony and Polar devices for example seem to implement overall solid security concepts, satisfying almost all points on our list. And even the point not explicitly fulfilled – the option of deactivating Bluetooth completely – is alternatively solved in an adequate way. On the other side the Acer tracker seems to have no security concept at all. The flaws in this case were so significant that we were able of completely exploiting almost all aspects of the tracker, including data theft and manipulation of some data and tracker functionality within just a few days of analysis. The apps and trackers in between primarily suffer from easily avoidable flaws like missing code obfuscation, unnecessarily detailed log statements and inconsistent final release checking. All of these points make a theoretical reverse engineering so much easier even for low-skilled attackers and therefore should be avoided at all costs – especially because their realisation does not seem to imply significant effort.

Concluding, we can state that this analysis is just a start up and that there are lots of more things to investigate. At first a more detailed look at the devices and apps is necessary to check for possible vulnerabilities regarding self-made apps which can allow for data manipulation or even forgery. Even the communication with the online cloud services with self-made, non authentic apps is within the bounds of possibilities for at least some of the trackers. For the use by insurances and similar instances, integrity and authenticity of the fitness data is the absolute priority and therefore has to be checked for. Some of the apps ask for a lot of permissions on the smartphone, some of them with quite doubtful necessity. How do the apps use the private data acquired with these permissions and what information flows to which partner network like Google Fit? These and other questions are still to be answered in future work.

Bibliography

[Barcena et al., 2014] Barcena, M. B., Wueest, C., and Lau, H. (2014). How safe is your quantified self? Online. last access January 27th, 2015. Available from: http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/how-safe-is-your-quantified-self.pdf.

[Margaritelli, 2015] Margaritelli, S. (2015). Nike+ FuelBand SE BLE Protocol Reversed. Online. last access April 30th, 2015. Available from: http://www.evilsocket.net/2015/01/29/nike-fuelband-se-ble-protocol-reversed/.

[Unucheck, 2015] Unucheck, R. (2015). How I hacked my smart bracelet. Online. last access April 30th, 2015. Available from: http://securelist.com/blog/research/69369/how-i-hacked-my-smart-bracelet/.