Evaluation of the security of smartwatch communication

Harm Dermois, James Gratchoff, Florian Ecard, Master SNE, UvA

December 2014

University of Amsterdam

Contents

1 Introduction

With the uprising of IoT (Internet of Things), more and more wearable devices start to get connected. One of these wearable device is the smartwatch. Smartwatches are used to display informations or notifications coming from a phone through a bluetooth communication. It can also be used to collect data.

With all new technology comes new security concerns. Security issues are even more relevant in the case of small battery reliant devices. As encryption and security can be very computation intensive, this will take huge toll on these devices. The aim of this project will be to investigate how secure the communication between a smartwatch and a smartphone is, and try to find solutions or alternatives to the problems found in our investigation.

2 Research Question

Information exchanged between smartwatches and smartphones are private and need good security in order not to leak sensitive information and be sure of the integrity of the data. The main mean of communication between smartwatches and smartphones is bluetooth. The use of low-power communication makes it more likely that there will be security issues with the communication and that encryption might not be set up as it is asking a lot of computation resources.

How secure is the bluetooth communication between smartphone and smartwatches? Is it possible to eavesdrop any data exchanged during this communication? Is it possible to change the content of the data exchanged? Is it possible to take control of a smartwatch and/or the smartphone?

How the vulnerabilities found (if any) could be addressed?

3 Ethics

From an ethical standpoint there will be no major issues for the experimentation that will be conducted. The equipment and the data used will be the authors/university property. The authors will not eavesdrop other people devices that are not involved in the experimentation. The publication of our paper could bring ethical issues as the smartwatch market is expanding and indeed if the authors come up with security flaws they could be used in an evil way by others. That is why the paper produced will evaluate the ethical problems brought by the research.

4 Literature Review

4.1 Bluetooth

4.1.1 Introduction

Bluetooth is a wireless communication technology created in 1994 by the mobile telecommunication company Ericsson. Bluetooth was designed for low-power consumption devices such as sensors, mobile phones, etc. Nowadays, more and more devices use Bluetooth mainly due to the uprising of Internet of Things (IoT) and small battery reliant devices.

4.1.2 Bluetooth communication

Bluetooth operates in the 2.4 GHz frequency band and uses frequency-hopping spread spectrum. This FHSS means that each packets from a Bluetooth communication is transmitted on one of 79 channels having a bandwidth of 1MHz each.

It is needed to describe older versions as well.

Currently, the latest version of Bluetooth is 4.1 that is known as Bluetooth low energy (BT-LE). As its name says, BT-LE was designed to reduce power consumption while providing the same communication range and speed that were used in the previous versions.
The case study in this project is however not using this latest version, but the Bluetooth 3.0 + HS (High Speed), this version is the first one providing a high data transfer speed that can go up to 24 Mbit/s. It is able to provide such a speed by using the famous 802.11 wireless protocol, it uses a "go all out" policy which first uses normal Bluetooth communication for pairing process and when a file is wanted to be exchanged or whatsoever then it swaps to the wireless in order to provide this speed.

Bluetooth is also using adaptive frequency-hopping spectrum (AFH), used to avoid crowded frequency in the channel spectrum.

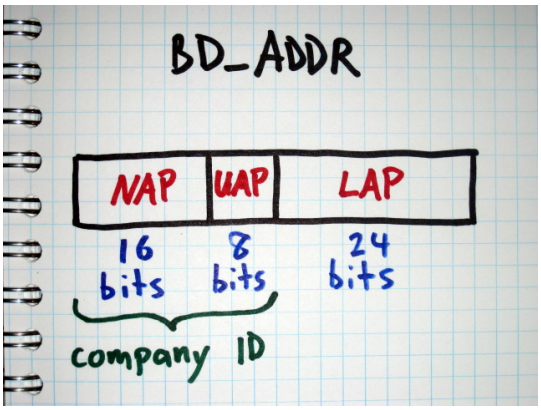
Device ID

The BD_ADDR of a device is composed of 48 bits. This address is divided into three parts, the first one being called the UAP (Upper Address Part) which is 8 bits long combined as well as the Non-significant Address Part (NAP) being 16 bits long form together the manufacturer ship, or company_id. The last part being 24 bits long is the Lower Address Part (LAP) almost uniquely identifying a device, this one is the one that will be used in the first stage of sniffing in order to decode the captured packets (This will be further explained later on).

4.1.3 Overview of Bluetooth security

The user sets the basic security in bluetooth. It has the choice between three different modes:

- Silent: In this mode, the device will not initiate any connection. The only thing it does is monitoring the traffic.



- Public: The device is discoverable by anyone around it having its bluetooth activated.
- Private: The device will in that case only answer to device that it already paired with, thus making it (theoretically) only discoverable to known

A bluetooth device can implement four different security modes:

- Non Secure: As its name suggests there is no security.
- Service-level enforced security mode: It establishes a non secure ACL (Asynchronous connectionless) link between the two devices willing to communicate. In order to introduce optional encryption, Authentication and authorization, a request has to be made via L2CAP (Logical Link Control and Adaptation Protocol) connection-oriented or connection-less channel.
- Link level enforced security mode: Once the ACL links are done then security procedures are initiated.
- Service-level enforced security mode: This one is very similar to the second mode except that it introduces SSP (Secure Simple Pairing) and this is compatible only with bluetooth versions from 2.1 + EDR.

Pairing process

SSP has been implemented in order to overcome the weakness of the 4 digits pass at pairing time (only about 10.000 possibilities max). Indeed, using an elaborated bluetooth sniffer would allow to almost instantly find these digits and then bypassing the security in addition to make the pairing process easier than it used to be.

The fact of pairing two pairing is actually to create a shared key, called the Link key. It will be used to authenticate and encrypt the data of two devices. In order to do so, there are two possibilities: LMP-pairing (4-digits/pincode manner) or SSP.

Once this done, the two devices can store their mutual key in order to use it later for a re-pairing, making the connection much faster.

In LMP-pairing, the only thing that is not transmitted through the air is the 4 digits PIN code. The pairing process in this case works as follow: The initiator will first generate a 16-bytes random number and then send it to the other device. Once received, both users will put in their PIN code which they will use to create an initialization key, the later will be converted into the actual LMP key. The two devices have to authenticate each other with their respective LMP key.
A software created by Ellisis [XXX] allows an attacker that listens to the traffic to guess with the given information the 4 digits, thus making LMP useless.

SSP was then created, fixing the above vulnerability even though we can still find this mechanism in many devices for backward compatibilities. SSP is a new manner of generating a link key (and user numbers such as PIN codes can still be used) using Elliptic Curve in order to create in this situation a much bigger random number now setting the number of possible Link keys to 2^{128} which is far too big to be brute forced, nowadays anyway.

In order to create a public/private key pair, SSP uses the famous Diffie-Hellmann algorithm with a 192-bits random number. The public key is transmitted over the air and is seeable by anyone, the private key is however kept secret.

Using two given key pairs A and B, there is a function that allows to result in a DHKey and only the two devices should be able to find it out. This function is F(Public A, Private B)=F(Public B, Private A). This very DSKey will be used in order to create the link key in between the two devices. From there, the rest of the pairing process is similar to the LMP-pairing.

Protection against MiTM with SSP

In order to be protected against MiTM attacks, it can use the user (e.g. to compare two numbers) or Out Of Band (OOB) channel (NFC for example, used to check the integrity of a checksum), these two cases are two of the four association model of SSP.
Let's define these models:
The numeric comparison association model which is the first case above, asking the user to compare numbers. The Passkey Entry association model which is used when one of the device has imput capabilities but no displays, in this case only the user with input capabilities will answer the 6-digit pass code. Finally, if one of the device has neither input nor output, and that OOB can't be used, then the JustWork association model is used, which is the weakest model as it simply asks the user to accept the connection.

In SSP there are 6 different phases: - Capabilities exchange: discovering devices or re-pairing ones will exchange their input/output capabilities. - Public key exchange: Compute the private/public key using Diffie-Hellmann and give to each other their public keys.
- Authentication stage 1: The protocol will depend on which of the four association model has been chosen. The aim of this stage is to be sure that nobody can eavesdrop the connection. To do so, the devices exchange nonces (random num-

bers) and commitment to them before checking the integrity of the checksum using either OOB or the user himself as discussed earlier.
- Authentication stage 2: There, the devices have exchanged their data and the integrity of each other is verified.
- Link key calculation: Using the previous steps, the Link key can be calculated (the DH key and nonces). It also needs the BD_ADDR.
- Authentication and encryption: Now the encryption keys can be generated in order to encrypt all the data that will be exchanged between the two devices.

While BTLE uses AES, the older bluetooth versions up to the 3.0+HS are using SAFER+ (Secure and Fast Encryption Routine) in order to provide authentication and key generation.

The information provided in this subsection are based on K. Haataja et al. book [XXX] and wikipedia [XXX].

4.1.4 Known Attacks on bluetooth

4.2 Ubertooth

4.2.1 Motivating the packet sniffing problem

Bluetooth technology is harder to sniff compared to other wireless protocols such as 802.11 that already have solutions for promiscuous packet sniffing. Because of howthe way Bluetooth works it is harder to tap the communication between two devices. The devices need to pair before real message are being send. Normally you can only see the discovery messages with are used to find out which Bluetooth devices want to be found. Another problem is that Bluetooth devices can choose to stay hidden. Due to frequency hopping it is hard to keep track of a device and listen to packet exchanged. However with the Ubertooth project it is now possible. To do so 3 parameters need to be known prior to be able to retrieve any information, the Lower Address Part (LAP), the Upper Address Part (UAP) and the clock CLKN that is the upper 26 bits of the CLK27 of the masters clock. The process to find this parameters is explained later on.

4.2.2 The Ubertooth project

Mike Ossman's project started in 2010 under the name of Ubertooth is dedicated to create and build a open source hardware that is able to follow and sniff Bluetooth communication. The latest release (Ubertooth 1) is capable of sniffing communication by identifying the right address and the right clock. The following sections are explaining how this is done.

4.2.3 The Ubertooth Tools

The Ubertooth comes with some tools to help you sniff packets. This is done by the host code which is provided by project. In the host code there are a few functions. The main one is $ubertooth - rx$ this is the tool that we will be using the most. $ubertooth - rx$ is the tool that does all the hard stuff. It listens to

all the packets send on the bandwidth and displays them. If you are using the default settings it will try to look for all the UAP of the devices. If you home in on a single device you want to track it will do more. It will then use the address it is following to find the clock and then decrypt the package. This package will then be printed with all the data inside. This is exactly where what we are looking for.dere is also a Kismet plug-in available which does the same as $ubertooth - rx$, but displays it in a better way. This also makes it log files in a pcap format which can then be used to analyse in wireshark.

4.2.4 Finding the UAP

The UAP (Upper Address Part) is the important to do anything interesting with Bluetooth. For instance it used to determine the hopping sequence that is used for the communication between two devices. It is 8 bits of the 48 bits BD_ADDR (Bluetooth Device address). The LAP is transmitted in plaintext so it not hard to find as it is present in all the packets. The First part is the NAP(Non-significant Adresss Part). This part is ignored because it is not needed for the initial communication. Because the UAP is only 8 bits it can be brute forced. This is not a good way to find it. Brute forcing is detectable and it will only work when the master is a connectible state. The Ubertooth is made to be a passive sniffer so this method is not used. One of the techniques used to determine the UAP is by using the HEC(Header Error Check). This is used to check if the header is received without errors. With this it is possible to determine what the are the unknown bits which will be the UAP. This header is send with each packet so it makes it easy to decode. The only problem is that it is XOR with a pseudo-random stream. To try and decode this is called "whitening". There are 64 possible streams it will use and this depends on which clock is used. It will then generate 64 candidate UAP with each of the 64 streams. Sometimes the packets will have a CRC that can be use to check if the right stream is used to decode the packet. Another method is to perform a series of sanity checks on the packet format. The packet type can then be unwhiten. If the packet type is known some information can be derive such as the packet length. These information can confirm or deny a possible match. The problem with this is that false negatives can happen and it can eliminate UAP streams which are valid. This can happen, because the data that is decoded will be of wrong packet type and it will make the wrong conclusion on whether to or throw away this possible clock. This will lead to not finding any UAP for the master and the process will have to be restarted in order to find it.

4.2.5 Finding the right clock

4.3 Sony Smartwatch SW2

4.3.1 Sony smartwatch 2

The smartwatch that was chosen to do the project with is the Sony smartwatch 2. It is a cheap watch and it is programmable. The SDK has recently been

updated and is easy to use. We have used the SDK to quickly make an app. There are many examples which are easily installed. These examples were used to generate traffic for the ubertooth to detect. This sped up the process of finding targeting and decoding the packets. All the Sony devices use an app called the smart connect. This app keeps track of all the devices that want to make contact with the phone. It also keeps track of the apps that are installed on the watch. Whenever the pairing is stopped between the phone and the watch all the "installed" app are removed from the watch until they are paired again.

5 Methodology

method

6 Results

result

7 Discussion

discussion

8 Conclusion

conclusion

9 Future Work

fw