Extracting the painful (blue)tooth

Matteo Beccaro
Matteo Collura

27 March 2015

Contents

In this paper we're going to analyze a logic vulnerability which allows an attacker to gain control over an Android device, with version prior 5.1 and SmartUnlock enabled.[1] The whole process has been tested against Android 4.4, Android 5.0, Android 5.0.1, Android 5.0.2 and Android 5.1; using two different smartphones, a Nexus 5, a Galaxy Note 3 (I9005), and a tablet, a Nexus 7 2013 (WiFi Only).

1 The SmartUnlock Feature

Let's see what the SmartUnlock is. Officially introduced with Android 5.0 it allows to unlock the smartphone without any user interaction, even if he sets a PIN, a passphrase, a sequence pattern or facial recognition, if at least one of the following conditions apply:

- The smartphone is within a certain location.
- The smartphone is in range of a previous saved NFC tag.
- The smartphone recognizes the face of the owner, which must be previously saved.
- The smartphone is in contact with a body.
- A previous enabled bluetooth device is connected to the smartphone.

I'm going to give a brief overview of all three possiblities, focusing on the last one, in which the vulnerability lives.

[1] At the time of writing the latest version bluetooth Android API are still vulnerable, which means that if you use a third part application the bypass is still possible.

1.1 Location Unlock

First of all the user must set a location as Trusted when he is in that location. Moreover to work this function must have a data connection enabled. It simply checks, using both data/WiFi and GPS, the user location, if it is within a range in the Trusted zone, the lockscreen is bypassed.

1.2 NFC Unlock

Again, the user must set a tag as Trusted, and then every time the smartphone can read that tag the lockscreen is bypassed. Android automatically warns the user when he tries to use this feature, as an NFC tag, in most of the case, can be easily cloned or manipulated. It is important to note that only the UID of the tag is checked, in case of Ultralight tags it means 6 bytes of entropy, (2^{48} possible values to bruteforce.).

1.3 Face Recognition Unlock

This feature allows the user to unlock the phone without any user-interaction if it is able to verify through the front facing camera that the user is the actual owner of the Android device.

1.4 Body Recognition Unlocl

This feature allows the user to unlock the Android device as long as it *on* the user i.e. hands, pockets, etc. This of course is secure since the Android devie cannot recognize if the body is the owner's.

1.5 Bluetooth Unlock

This may be the most interesting and most used function of all the above. The user set a bluetooth device as Trusted, and from now on every time that device is linked to the smartphone the lockscreen is bypassed. I personally find this feature very usefull for example when I'm driving and my smartphone is linked to the car's infotaiment: I can search location via Google Now, etc without entering the PIN or passphrase which can be distractive. It is important to note that before being able to set a device as Trusted the smartphone must be paired with that device, which means as we will see shortly, that it and the device share a LinkKey (LK).

2 The Bluetooth Protocol(s)

Let's go a bit deeper in details about how bluetooth works.[2] The bluetooth is a technology which allows two devices to communicate wireless if they are within a certain range; this range depends on the transreceiver of each device.

- Class 1 devices have a range of 100 meters and a transmitting power of 100mW
- Class 2 devices have a range of 10 meters and a transmitting power of 10mW
- Class 3 devices have a range of 1 meters and a transmitting power 1mW

The specifications have grown a lot in the last years and for semplicity they will be divided in three main groups:

- Bluetooth Legacy
- Bluetooth EDR
- Bluetooth LE[3]

For all the following sections the security aspects will be highlighted. A little note, in the following document there will be often references to UAP, LAP and NAP, those are section of the bluetooth MAC address:

$$\overbrace{00:11}^{\text{NAP}}:\overbrace{22}^{\text{UAP}}:\overbrace{33:44:55}^{\text{LAP}} \tag{1}$$

2.1 Bluetooth Legacy

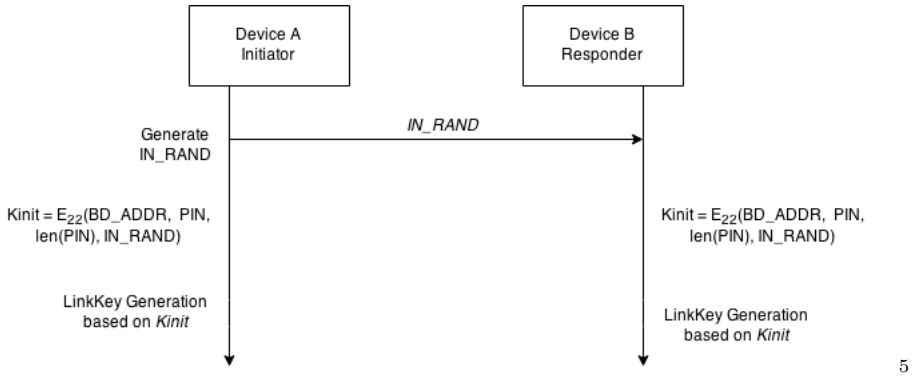
The first versions of bluetooth guarantees a very poor security level. Up to the version 2.1, the communication could be established also in an unencrypted way. It was anyway difficult to eavesdrop a communication because in version 2.0 was implemented the AFH, Adaptive Frequency Hopping It was mainly developed to reduce noise since it changes the communication channel around 1600 times per second, and the bluetooth has 79 channels, 1MHz of bandwitdh each. That means 79MHz of spectrum to record. Some attacks have been developed to be able to follow the jump series, but they require some prior knowledge: the hopping sequence is determined using two values, the clock value and the BD_ADDR[4].

[2] It is not possible to cover all bluetooth functionalities in just one paper, please refer to the official specifications for further details – `https://www.bluetooth.org/DocMan/handlers/DownloadDoc.ashx?doc_id=286439`

[3] This specification implements a whole new protocol, in PHY and Link layer, but it will be discussed more in details later

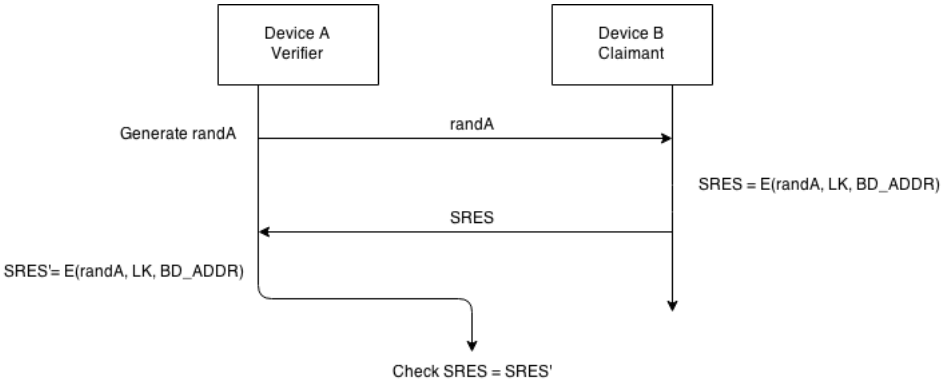
[4] The bluetooth device mac address

The pairing process is based on a PIN code, which is shared between the two devices:



[5]

Device A generates a random number, IN_RAND, and transmit it to Device B. Then using the PIN provided by the users, both the devices calculate *Kinit*, a cryptographic hash using these parameters and the BD_ADDR. If those hashes correspond, then the pairing process is successful, and a LinkKey (LK) is generated from this hash. The flaw here is pretty easy to see: if it's possible to eavesdrop a pairing process both the random numbers and the BD_ADDR are known to the attacker, who can bruteforce the PIN code in order to get the correct LinkKey (LK).
The authentication on the other handis made in the following way:



[6]

- In the authentication process the Verifier generates and sends to the Claimant a random value, *randA* .
- Device B evaluates function *SRES*, based on *randA*, *BD_ADDR* and *LK*; then sends the result to Device A.
- Finally, the Verifier evaluates *SRES'* at the same way, and checks if the result is equal to *SRES*.

[5]The BD_ADDR is the Responder's one
[6]The BD_ADDR is the Claimant's one

2.2 Bluetooth EDR

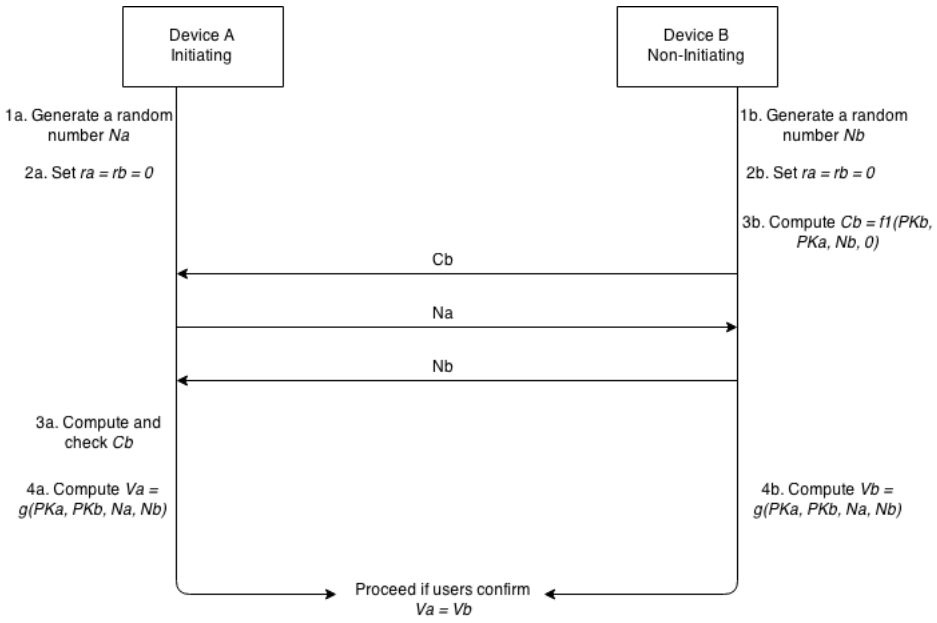
Among various speed improvments these new specifications of the bluetooth protocols includes a new method for pairing: the so called Secure Simple Pairing. This method is divided in five different steps:

1. Public Key Exchange
2. Authentication Stage 1
3. Authentication Stage 2
4. Link Key Calculation
5. Encryption

on the type of device we are trying to connect the bluetooth EDR supports different types of pairing:

- Numeric Comparison
- Passkey Entry
- Out of Band (OOB)

We focus on the *Numeric Comparison* method, in which the most important phase is first stage of the authentication, which works as follows:



1a. & 1b. After the exchange of the public keys[7] device A and B select a random number, respectevely *Na* and *Nb*.

2a. & 2b. Both the devices set *ra* and *rb* to 0.

3b. Device B computes the function *Cb* and sends it to Device A along with *Nb*.

3a. Device A sends its random number, *Na*, to Device B, then computes and verifies *Cb*. If this fails the pairing procedure aborts.

4a. & 4b. Device A and B evaluate respectevely the function *Va* and *Vb* and prompt the users to check if the results matches. If so, the Authentication Stage 1 is completed.

2.3 Bluetooth LE

The Bluetooth LE has been introduced in version 4.0 of the bluetooth specification document. From a security point of view the pairing methods are similar to the EDR modes[8], with the difference that *Just Works* and *Passkey Entry* modes don't have MiTM protection due the fact they are not using Elliptic Curve Diffie-Hellman (ECHD).

[7]The two devices, A and B, exchange their DH keys, called respectively PKa and PKb, during the first step of the Secure Simple Pairing

[8]For a scheme please refer to this list

3 Android Lockscreen Bypass

3.1 SmartUnlock Bypass

As seen before the SmartUnlock feature allows to unlock the Android smartphone without inserting the PIN/passphrase code. The problem is that Android doesn't check if a device has the proper LK before setting it as connected. This connection then will be dropped as soon as some data are requried to be exchanged, and in that case Android recognize that the LK is not set, or it is wrong, in the target device. That means that it is possible to connect to the Android device even without the correct LK for a little amount of time. There is still a problem to be solved before starting the connection to the Android device and unlock the lockscreen: the BD_ADDR of the target device is required. There are two possible solutions, first is to bruteforce the BD_ADDR, second one is to eavesdrop a communication between the target device and the Android device and retrive it from there. The bruteforce solution has 48bits of entropy, indeed only the LAP and UAP of MAC address are checked[9], therefore the entropy is dropped to 32bits, moreover we can parallelize the attack using several bluetooth dongles.[10] The second attack is faster but requires to be near the target during the communication between the Android device and the target device. Then it is possible to intercept the communication and retrive the BD_ADDR using for example an ubertooth[11].

3.2 API Vulnerability

Even if the SmartUnlock feature has been fixed in thelatest Android release, Android 5.1, the bluetooth API are still vulnerable, and doesn't allow to check properly if a connected device has or not the correct LK. It is possible to cheat any application which uses those API to believe that the device is authorized using the same attack shown in the previous section.

String	ACTION_ACL_CONNECTED	Broadcast Action: Indicates a low level (ACL) connection has been established with a remote device.
String	ACTION_ACL_DISCONNECTED	Broadcast Action: Indicates a low level (ACL) disconnection from a remote device.
String	ACTION_ACL_DISCONNECT_REQUEST	Broadcast Action: Indicates that a low level (ACL) disconnection has been requested for a remote devices, and it will be soon disconnected.

[9]The bluetooth protocol makes no difference between the address: 00:00:11:11:11:11 and the address 11:11:11:11:11:11

[10]A compatible bluetooth dongle can be found very cheaply on eBay for around 1-2 USD

[11]`http://ubertooth.sourceforge.net`

3.2.1 PoC

We created an Android application which, using the official bluetooth API, updated to latest release, checks when a bluetooth device is set by the system as *Connected* and/or *Bonded*. This allows to check if future version of those API are still vulnerable.[12] A device is seen as *Connected* if a link between the device and the smartphone is created, while it is considered *Bonded* if the smartphone has saved in the file located in */data/misc/bluedroid/bt_config.xml* a LK for that device. The PoC application, and the source code, is available on GitHub[13]

```
if (BluetoothDevice.ACTION_ACL_CONNECTED.equals(action)) {
   if( device.getBondState() == 12 ) {
     bondedState = "Bonded";
    }
    else {
     bondedState = "Unbonded";
    }
    deviceText.setText("Name: "+device.getName()+"\nAddress:
        "+device.getAddress());
    statusText.setText("Connected"+"\nBonded: "+bondedState);
```

3.3 Reducing the entropy

As seen until now it is required to bruteforce 2^{32} possible addresses in order to be sure to have found the correct one. This makes the bruteforce attack nearly impossible. But, Android has an other logic vulnerability which allows to reduce this entropy to 2^8 possible addresses, which means that it is possible to bruteforce and find the correct address in just a couple of minutes with an automatic tool, or around an hour if you do it manually. This is possible because when you soft-reset the bluetooth on Android it starts searching for known bluetooth devices, sending out the LAP part of the bluetooth address of those devices. It will broadcast beacons, which can be easily eavesdropped by an ubertooth. Once the LAP is known only the UAP is missing, which means 8bits.

[12] At time of writing, the Android SDK 22 is vulnerable.
[13] `https://github.com/bughardy/(blue)toothmark`