Security Analysis of Bluetooth Technology

Daniel Filizzola
danifili@mit.edu

Sean Fraser
sfraser@mit.edu

Nikita Samsonau
samsonov@mit.edu

Abstract

With the increasing popularity of wireless communication systems, Bluetooth has become one of the most widely used short range wireless protocols. Unfortunately, due to the nature of wireless communication networks, Bluetooth has security vulnerabilities particularly through eavesdropping. Despite newer and more secure versions of Bluetooth being released, older versions such as Bluetooth 4.0 and 4.1 are still widespread all over the world. After thoroughly exploring the current Bluetooth security model and reasons for potential vulnerability, this report performs a comparative analysis of different Bluetooth security attacks, extending them and applying them to readily accessible devices, and offering countermeasures. Based on our results and discussion, it is clear that Bluetooth is a widespread technology with significant security vulnerabilities in the real world today.

1 Introduction

Wireless communication systems and their interconnections by networks have become increasingly popular in recent years, particularly with the surge of interest in the Internet of Things (IoT). The most common wireless communication systems use Radio Frequency (RF) waves, which can penetrate objects and operate without direct line of sight between devices. Despite being easier to use than wired communication, this makes the communication more susceptible to eavesdropping or disruption.

Bluetooth [1] is a short-range wireless data transfer that operates in the 2.4GHz frequency range with multiple data transfer rates possible with real-time two-way rates up to 24Mb/s. Nowadays, Bluetooth devices are seamlessly integrated into our daily lives, in the form of headsets, smart phones, mice, keyboards and portable speakers, and are widely used all over the world. Furthermore, hands-free voice communication devices are becoming increasingly more popular in cars for safety reasons or homes for convenience. Given that Bluetooth devices are so widespread, it is often the case that some security principles are neglected and vulnerabilities are exposed.

Despite the Bluetooth Special Interest Group releasing newer versions of Bluetooth such as version 5.0 (currently compatible with newer technologies such as Apple AirPods), there is still a huge number of devices in use that use older versions of Bluetooth, such as Bluetooth Smart / Bluetooth Low Energy in version 4.0 and 4.1 [2]. In fact, it is estimated that there are over 4 billion Bluetooth Low Energy (BLE) enabled devices in 2018 (using version 4.0 or 4.1) [3].

In this report, we illustrate the fact that Bluetooth is a widespread technology with real security vulnerabilities and implications for their practicality in the real world. By engaging with previous work done in this area, we explore various potential vulnerabilities in Bluetooth networks and apply extensions of various attacks to arbitrary Bluetooth devices and evaluate the results. In one case, we find that the results are critical in which we eavesdrop and decrypt computer mouse location data, thereby enabling a third-party to extract sensitive information.

In Section 2, we give a general overview of Bluetooth security. In Section 3, we explore why there is potential for the Bluetooth medium, protocols, and parameters to exploited. In Section 4, we introduce various Bluetooth attacks done in previous work, as well as the two specific attacks we focus on in Section 5 and Section 6. These sections explain in detail our comparative analysis of the attacks and our extensions to them. We discuss the results of our attacks and how practical they are in Section 7. Finally, in Section 8 and 9 we finish with suggestions for future work in this area and our concluding remarks.

2 Overview of Bluetooth Security

This section provides an overview of the security goals and mechanisms as provided by the Bluetooth standard. We aim to illustrate the strengths and limitations of the current standard of Bluetooth security.

Bluetooth devices that communicate with each other form a *piconet*. A piconet *master* is a device that initiates a connection and there can be up to seven *slaves* in the piconet. All communication has to go through the piconet master however. Slave devices (e.g. headset, mouse, keyboard) connect with master devices by a pairing process as shown in Figure (2) below in Section 2.2.2.

2.1 Security Goals

There are five basic security goals and principles specified by the Bluetooth standard [4]:

- **Authentication:** One of the goals of Bluetooth is to be able to verify the identity of communicating devices with a unique Bluetooth address. Bluetooth does not provide native user authentication.
- **Confidentiality:** Another goal is to prevent the disclosure of information caused by eavesdropping by ensuring that only authorized devices can access and view transmitted data.
- **Authorization:** The Bluetooth standard sets out to allow the control of resources by ensuring that a device is authorized to use a service first.
- **Message Integrity:** Messages sent between two Bluetooth devices should not be modified in transit.
- **Pairing/Bonding:** The last goal of the Bluetooth standard is to create one or more shared secret keys between two devices and storing them for future use in subsequent communications.

2.2 Security Mechanisms

Bluetooth implements the above security goals with several mechanisms. Some related aspects of the Bluetooth protocol to provide mechanisms for the security goals as outlined below.

It is worth noting that for this report the difference between Bluetooth and Bluetooth Low Energy (BLE) / Bluetooth Smart / Bluetooth 4.0 protocol. The key difference is the low power consumption of BLE, and the result of this is that is a different protocol with different security goals, mechanisms and vulnerabilities. Older and newer versions such as Bluetooth 4.2 exist, but many of the vulnerabilities in the mechanisms are specific to one version. General aspects of the Bluetooth protocol are described below as security mechanism.

2.2.1 Bluetooth Device Address

Every Bluetooth device has a unique 48-bit number called a Bluetooth Device Address or `BD_ADDR`, used to identify the device. The structure of the address is shown in Figure 1. This is similar to a MAC Address for Ethernet and Wi-Fi networks, however there are some key differences. Unlike a MAC Address, the `BD_ADDR` is used throughout the Bluetooth protocol, for identity, authenticity, and low-level communication. For example, all devices transmit using the master's `BD_ADDR`.

As Figure 1 illustrates, the `BD_ADDR` is split into three parts:

- the 16-bit Nonsignificant Address Part (NAP)
- the 8-bit Upper Address Part (UAP)
- the 24-bit Lower Address Part (LAP)

The first three bytes (NAP and UAP) refer to a `company_id` or the manufacturer assigned part, and is publicly available. The last three bytes (LAP) are the `company_assigned` identifier. Because the LAP is widely used throughout the Bluetooth protocol, it is important that it remains private. However, when Bluetooth devices are in Discoverable Mode, the `BD_ADDR` advertised, and even when not in this mode, Bluetooth packets are sent with the LAP of the device in plaintext, or if in a piconet, the LAP of the master. Knowledge of the LAP and UAP allows for the passive monitoring of packets and potential for attacks depending on the encryption used.

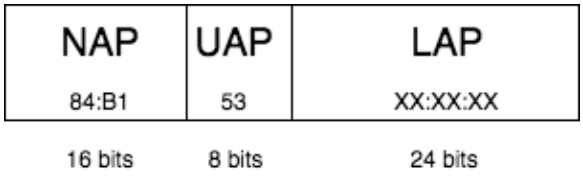

Figure 1: The format of the 48-bit Bluetooth Address

2.2.2 Pairing

In order to establish a Bluetooth connection, the involved devices need to start a pairing protocol in which they create and store the link keys that will be used for later data encryption (2.1).

Bluetooth BR/EDR, the standard Bluetooth, utilizes Secure Simple Pairing (SSP) since Bluetooth version 4.0 [2].

Bluetooth Low Energy before version 4.2 uses LE Legacy Pairing. BLE devices with version 4.2 and beyond utilize the protocol LE Secure Connections, which is similar to the SSP protocol. It was introduced to provide more security against MITM and passive eavesdropping attacks [1].

2.2.3 Association Model

Bleutooth BR/EDR SSP and LE Secure Connections implement four association models: [4]:

- **Numeric Comparison:** Both devices' display show a randomly generated six-digits number. The users of these devices confirm that these numbers match by clicking either a "yes" or "no" button. If both users respond with "yes", the pairing process is initialized.
- **Passkey Entry:** The display of one of the devices shows a six-digit PIN and the other user of inputs the same PIN with a keyboard in order to start the pairing process. Alternatively, if both users have a keyboard, inputting the same six-digits PIN will have the same effect.
- **Just Works:** The involved devices immediately start the pairing process upon request.
- **Out of Band (OOB):** For this model, both devices need to implement a different wireless communication technology such as Near Field Communication (NFC). However, this method is the least common due to the hardware requirements.

LE Legacy Pairing does not provide Numeric Comparison, while still providing Passkey Entry, JustWorks and OOB. However, the implementation of JustWorks and Passkey Entry in LE Legacy Pairing is less secure than in LE Secure Connections and SSP.

The choice of association model depends on the input and output capabilities of the device, as shown in table 1.

Table 1: Table indicating the choice of association model given the input and output capabilities of the Bluetooth devices as described in [5]

.

Device A	Device B	Assoc. Model
DisplayYesNo	DisplayYesNo	Num. Comparison
	DisplayKeyb	Num. Comparison
	KeybOnly	Passkey Entry
	NoInpNoOut	JustWorks
DisplayKeyb	DisplayOnly	Num. Comparison
	KeybOnly	Passkey Entry
	NoInpNoOut	JustWorks
KeybOnly	KeybOnly	Passkey Entry
	NoInpNoOut	JustWorks
NoInpNoOut	NoInpNoOut	JustWorks

3 Reasons for Bluetooth Vulnerabilities

Security in Bluetooth networks depends on the security of the Bluetooth medium, protocols and parameters. Weaknesses in each of these aspects result in vulnerabilities in a Bluetooth network.

Bluetooth makes several assumptions about security. For one, it assumes that once a connection is established between two devices, it will remain permanently secure with the keys it has stored. Secondly, there is an assumption that short range provides some level of security since adversaries need to be in close proximity (typically 5-30m). Moreover Bluetooth authenticates devices, not users, so there is an assumption that all users on a particular device should follow the same security protocol.

In this section, we look at Bluetooth's vulnerability to eavesdropping and weaknesses in the mechanisms of Bluetooth's security model.

3.1 Vulnerability to Eavesdropping

Due to the nature of wireless RF communication, eavesdroppers are often not detected in a Bluetooth network. Unencrypted transmissions obviously make it easy for an eavesdropper to see the contents of any packets. However the eavesdropper has to be in range of the Bluetooth network.

Bluetooth packets consist of an access code, packet header and a payload. Since the access code and packet header are always sent unencrypted, even when encryption is used on the data and payload, an eavesdropper can always see general piconet information pertaining to the devices. Using this information, the eavesdropper could figure out the authorization levels of the legitimate piconet devices.

Furthermore, as explained in our attack below 6, low energy legacy pairing provides no passive eavesdropping protection. If successful, eavesdroppers can capture secret keys distributed during low energy pairing.

3.2 Potential Weaknesses

3.2.1 Encryption Mechanisms

The most significant weakness in the Bluetooth encryption mechanism is when 128-bit encryption cannot be used. When two devices communicate the parameters for encryption, the length of the encryption key is restricted

by the Bluetooth device that has the shorter maximum encryption key length. Another weakness occurs when the PIN inputted in the Passkey Entry or JustWorks association model is used to generate the link keys. This PIN can be brute-forced and used for replication of the link keys by a passive eavesdropping attacker. An exploitation of this weakness is demonstrated in Section 6.1.

3.2.2 Association Models of SSP

One particular weakness of the association models of SSP that we look at is the Just Works model. This pairing method provides no MITM protection between trusted devices, as explored in Section 5.

3.2.3 Device Configuration

The default settings of Bluetooth provide little security, as the device is set as discoverable and non-secure, meaning that an attacker can discover the `BD_ADDR` of it and perform various attacks.

4 Overview of Bluetooth Attacks

4.1 Known Attacks

- Da-Zhi Sun et al. [6] showed that the passkey entry association model in Secure Simple Protocol (SSP) of Bluetooth in the newest version 5.0 is vulnerable under Man-In-The-Middle attacks. These are possible when the host reusing the passkey or generates it by a non-random algorithm.
- Cope et al. [7] aimed to capture connection request packets and to view the encrypted packet connection from master to slave. They found limited success in some devices such as a Fit Bit, where they eavesdropped on packets in its connection to an iPhone.
- Das et al. [8] proposed two new device authentication and data transmission protocols based on the current security issues and limitations of Bluetooth.
- Ryan [9] intercepted and reassembled packets using an eavesdropping technique, in addition to an making attack against the key exchange protocol that rendered encryption useless to passive eavesdroppers.

4.2 Challenges

Previous work shows that Bluetooth sniffing is very hard. Unlike Wi-Fi, Bluetooth employs frequency hopping between 79 channels approximately 1600 times per second. Furthermore, before transmission, both the header and the payload for each packet are scrambled with data whitening in order to randomize the data. This is more of an inconvenience to sniffing as opposed to a security mechanism. Finally, to sniff any packet over a wireless Bluetooth network, the receiver needs to operate in promiscuous mode, receiving all packets it can read without any regard of who it was intended for. General purpose Bluetooth firmware and hardware general do not support this mode, so more specialized hardware is needed. Moreover this hardware is often rare and expensive. One particular solution we utilize is Ubertooth [10], as explored in Section 6.

4.3 Our Attacks

Our goal in this project was to do a comparative analysis of previous attacks and extend them to arbitrary Bluetooth devices. In general terms, the two main types of attacks we explore are Active Eavesdropping or a Man-In-The-Middle (MITM) attck, and Passive Eavesdropping or Sniffing attacks.

5 Active Eavesdropping Attack

One immediate piece of information to notice is that the `JustWorks` mechanism does not provide any protection from MITM attacks. The most obvious devices employing `JustWorks` that come to mind are headphones, which become a great target to attempt, since people use them for private communication. To identify a starting point for the attack, we analyze what happens when a user tries to connect to a headphones/headset device. Although Bluetooth (BT) Headphones class and Bluetooth headset class have drastically different specifications, they are used interchangeable in this paper, since they perform the same purpose.

For the scope of the project, we perform a very high level attack. The approach we used is to use a lower level only if it is absolutely necessary. As we will see bellow, this approach has brought valuable results with less required effort, allowing us to try more things, yet left us with some limitations and gaps. The results, however, are enough to show substantial flaws at the higher level Bluetooth design.

5.1 Device used

For this attack, we start with Bluetooth 4.2 Headphones, bought on Amazon[1], as our slave device. As claimed on the product page, the headphones support Bluetooth 4.2 and easy pairing with tablets. Upon connection to

[1] https://www.amazon.com/Headphones-TaoTronics-TT-BH07-Waterproof-Cancelling/dp/B06ZYX6Y1T/

a laptop, the headphones add two entries to the list of available sound playback devices: the first one is listed as `TaoTronics TT-BH07 Hands-Free` with a Headset type, and the other one is listed as `TaoTronics TT-BH07 Headphones` with a Headphones type. Later, we see how these two types reflect the broadcasted services.

The master device used was a `Samsung Galaxy S7 Edge`(Galaxy) smartphone. As claimed on the manufacturer's website [11], the device supports BLE 4.2.

For the MITM device, we use `Raspberry Pi 3 Model B` *(RPi)*. We've chosen this device for the following advantages: it supports Bluetooth LE 4.2 technology; it uses Debian-based OS, which allows us to have a more fine-grained control of Bluetooth than it would be on a Windows or Mac device, yet still providing a high level control; it's mobility can be useful in demonstrating the concept of attacks in public places, since it is not restricted to any location and therefore, can bypass the assumption that it's hard for an attacker to be close to the victim.

5.2 Pairing mechanism

5.2.1 User perspective

The first time the user has to connect to the headphones, he/she has to hold a button on the headphones, until the LED starts to indicate that the headphones are ready to pair. Then, he/she has to go to the Bluetooth settings and add a new device. On the Galaxy the device appears as `TaoTronics TT-BH07` with an icon of a multimedia device. If the headphones weren't put in the pairing mode, the device would not be displayed. After connecting to the headphones, the user can allow the headphones to be used as a media playback device and/or a call audio device. So, from the user prespective, the only ways to identify the headphones are the name, the icon, and the functions broadcasted, although the last once is not obvious.

5.2.2 Technical perspective

On the lower level of the process, the smartphone sends inquiry requests [12] to a number of different frequencies, then scanning from a reply on some of these frequencies. If a potential device is found, the device broadcasts some information like the name, MAC, class and broadcasted profiles. Since this is a higher level attack, the actual process of pairing is irrelevant, but it will be discussed later in the paper. We discuss the role of each one:

- **Name.** The name of the device is what the phone displays to the user. It does not have any other purpose besides helping the user to identify the device.

- **Class.** Class[13] is what dictates the icon of the device, or more generally, what kind of device it is. Using RPi, we were able to extract the class ID - `0x240404`, which corresponds to Headphones/Headset device.

- **MAC address.** It is the most important identifier, which is used by the software to keep track of other devices. As seen in UNIX systems, the pairing link keys are stored in the location associated with the device's MAC address [14]. In our first experiment, we'll see that Galaxy also uses it to identify devices in the list of discovered devices.

5.3 Setting up the spoofing device

5.3.1 Experiment 1: Mimicking visual identity

We can now setup the RPi to broadcast the same metadata as the headphones. To get control over Bluetooth on RPi, we use `BlueZ` - a common Bluetooth stack for Linux. We can use `hciconfig` to change the name and the class of device. We managed to change the MAC address of the bluetooth module by using `bdaddr`[15] utility. However, we will not do it for the first experiment. In addition to that, we need to make our RPi to appear as a NoInputNoDisplay device to enable `JustWorks` , which we can do using `bluetoothctl` utility.

```
$ hciconfig hci0 name "TaoTronics TT-BH07"
$ hciconfig hci0 class 0x240404
$ hciconfig hci0 piscan

$ bluetoothctl
# agent NoInputNoOutput
# default-agent
```

Now, using the Galaxy, we can lookup available devices. Indeed, now we see a media device name "TaoTronics TT-BH07". However, if the phone has already been paired with the headphones, this will appear as a separate entry. When trying to connect to the device, we see two obstacles. On the RPi, a prompt requiring a confirmation pops up, which is an example of the limitation we couldn't by pass within the scope of the project and deemed it irrelevant. However, the phone doesn't actually connect to the RPi, but only pairs, since there is no profiles being broadcasted. Yet, we've shown that it's possible to connect to RPi without MITM protection and visually tricking the user into connecting to the spoofing device.

5.3.2 Experiment 2: Mimicking services

The next step towards making the RPi to appear the same as the headphones is opening the same communication channels by broadcasting the same profiles. Profiles are defined by services that support the profile, where each service has an associated socket. So, we want to extract a list of broadcasted services from the headphones.

Extracting services We want to imitate the services, while being able to eavesdrop the incoming traffic, so we can save it, and so we can transmit it to the actual headphones. To do so, we used `PyBluez`, which is a Python library allowing access to Bluetooth resources on the RPi. It has methods to discover devices in the range and discover the services broadcasted.

With the headphones, however, PyBluez was not returning any services when asked to provide all services[2]. Instead, we could use `bluetoothctl` to lookup all broadcasted profile UUIDs. We can do so by pairing to the headphones and using `# info` *MAC_ADDRESS*, which displays a list of long UUIDs, where 4-digits represent a profile.

For each displayed profile, we use PyBluez to display the necessary services. In the end, we end up with 5 different services. In the end, we end up with 5 unique services, with the following profiles:

- Serial Port (Port 1)
- Hands-Free (Port 2)
- Headset (Port 3)
- AV Remote (Port 24) [Volume Control]
- Advanced Audio (Port 25) [Audio Sink]

Now, we can recreate the services using PyBluez (code in the referenced repository). We make a script that broadcasts the services in such a way that, as soon as a master device connects to the port, the script connects to the same port on the actual headphones. We do so, by running all the sockets in non-blocking parallel threads, seamlessly resending any received data to the opposite device.

When we run the script, we put both headphones and the phone in the pairing mode. We make the phone connect to the RPi, and RPi in turn connects to the headphones.

[2]Later, when trying to extend this attack to a BT mouse, we realized a flaw in this approach that might explain the shortcomings in the expected results. Specifically, `PyBluez` supports BD/EDR implementation and has only experimental support for BLE and, more importantly, GATT. Therefore, not all the services might have been visible by PyBluez

Our script shows us that the phone connected to only two of the open ports. Yet, we can see BT commands being exchanged in a non-encrypted form. This shows that our setup works as a MITM spoofing device.

```
    listening on port 25
    listening on port 23
    listening on port 3
    listening on port 2
    listening on port 1
    2 Accepted connection from ('8C:1A:BF:5B:A5:8D',
    2)
    25 Accepted connection from ('8C:1A:BF:5B:A5:8D',
    25)
    25 Received from master b'\x00\x01'
    25 Received from slave b'\x02\x01\x14\x08\x10
    \x08\x04\x08'
    2 Received from slave b'AT+BRSF=191\r'
    2 Received from master b'\r\nOK\r\n'
...
```

The only shortcoming, however, that we weren't able to intercept the sounds. Even though there is a socket corresponding for an audio sink, the audio is not being sent through it. Most likely, the sound itself is being sent via SCO socket, yet the setup wasn't straightforward and was deemed irrelevant, since the fact that we intercept the BT commands is enough to show the capability of MITM, and there is no reason to assume the same can't be done for the sound.

5.3.3 Experiment 3: Mimicking MAC address

Now, we can perform a more sophisticated attack. We install an additional bluetooth adapter to an old one (hci1 and hci0, respectively). By using the `bdaddr` utility, we set the address of hci0 to the address of headphones, and the address of hci1 to the address of the phone. We also adjust the script by initating the listening sockets from hci0 and the outgoing ones from hci1.

When we run the experiment what we see is, if the headphones are within the phone and are in the pairing mode, the phone is likely to connect to either one. This is not the problem for the outgoing socket, since our spoofing socket does not accept more connections, so it will only hear back from the actual headphones. So, with probability around 50%, we are able to be a man in the middle.

5.4 Forcing the user repair

The attack requires the headphones to be in a pairing mode and the smartphone to be attempting to connect to the headphones.

Even though people do occasionally repair the headphones, we can force them to do so, using the fact that the phone won't connect to a device if link keys don't match. Therefore, if we are able to put our spoofing device next to the phone prior the phone trying to connect to the already paired headphones, there is a chance that the connection will be rejected and the user will be tricked into initiating repairing with the headphones, making it a very practical attack.

6 Passive Eavesdropping Attack

A passive eavesdropping attack is the process by which a third party listens to the data being exchanged by two devices. As mentioned in the Bluetooth specifications [2], BLE devices with version 4.0 and 4.1 are vulnerable to this kind of attack. Even though BLE devices encrypt data by using AES-CCM cryptography, the key exchange protocols are still exploitable. This is not the case for BLE version 4.2 and beyond due to the introduction of ECDH (Elliptic Curve Diffie Hellman) key exchange, which is proved to be secure under this type of attack [1].

6.1 How the attack works

Before two BLE devices can initialize an encrypted session, they need to share a secret LTK (Long Term Key). Usually, this LTK is created only once and generated when the devices connect for the first time. This key exchange protocol starts with the devices sharing a TK (Temporary Key), which is a 128-bit value used as a key for AES encryption. Depending on the association model, TK will take different values [9]:

- **JustWorks**: The value of TK is just 128 0's.
- **Passkey Entry:** The value of TK is a 6-digit PIN, given by the users of the devices. Then, this value from 0 to $999,999$ is padded with 0's to get the 128-bit key.
- **OOB:** a 128-bit value exchanged out of bounds.

Then, the two devices start the pairing process by sharing the following messages [16]:

- Pairing Request command (*PREQ*) and Pairing Response command (*PRES*)
- Initiating device address type (*IAT*) and Initiating device address (*IA*)
- Responding device address type (*RAT*) and Responding device address (*RA*)

After sending and receiving these unencrypted messages, the master and slave device generate *MRAND* and *SRAND* respectively. These two numbers are pseudorandom and not shared initially.

Then the master computes:

$$MCONFIRM = c1(\text{TK}, MRAND, PP)$$

where c is a function uses the public parameters $PP = \{PREQ, PRES, IAT, IA, RAT, RA\}$ and AES-128 encryption with TK as the key. Similarly, the slave computes *SCONFIRM* using *SRAND* [2].

After computing these values, they send *MCONFIRM* and *SCONFIRM* followed by *MRAND* and *SRAND*. Then, the master and the slave recompute *SCONFIRM* and *MCONFIRM* locally and check that it matches the value that were sent to them. This step is done to confirm that both the master and the slave have the same TK [16].

Once the master and slave confirm that they have TK, they compute the short term key

$$\text{STK} = s1(\text{TK}, SRAND, MRAND)$$

where $s1$ uses *MRAND* and *SRAND* and AES-128 encryption with TK as the key [16].

Finally, this STK is used for exchanging the LTK, which will be used for future communication and encryption [9]. This process is synthesized in Figure 2.

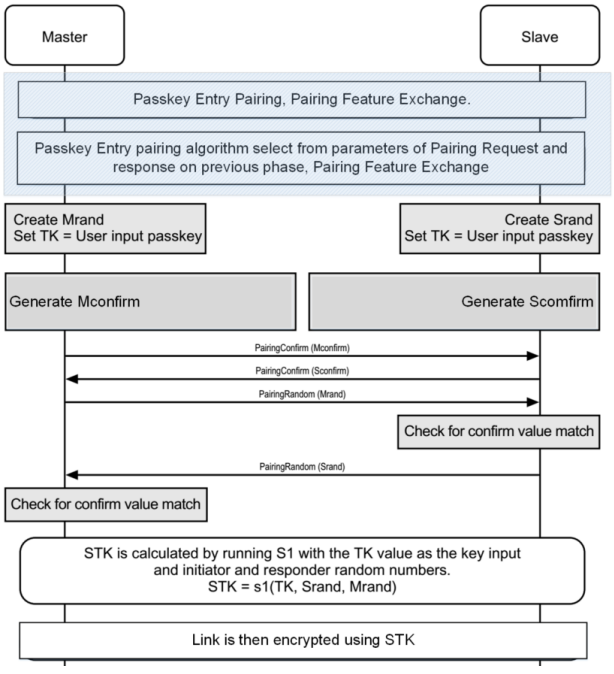

Figure 2: Overview of the STK and LTK generation. This figure was extracted from [16]

.

This pairing protocol called LE Legacy pairing proves to be vulnerable for the association models JustWorks and Passkey Entry. If a passive eavesdropper listens to all the messages in this pairing protocol, the attacker can simply brute-force the 1,000,000 possible values of TK until it gets a key that correctly computes *MCONFIRM* and *SCONFIRM* to get SKT. Then, the attacker can use the obtained STK value to compute the LTK [9].

6.2 Countermeasures

Note that the attacks relies on the fact that there are not that many possible values for TK, which ranges from 0 to 999,999. Thus, this protocol is secure in the OOB association model, where TK takes a random 128-bit value. However, OOB is not commonly implemented since it requires special hardware [9] .

In order to prevent this attack on association models such that JustWorks and Passkey Entry, BLE 4.2 introduced a ECDH key exchange protocol called LE Secure Connections. This protocol is similar to the SSP used by Bluetooth devices since 4.1 and it is secure against passive eavesdropping attacks [1].

6.3 Setting up the attack

We decided to replicate this attack on a BLE 4.1 Logitech MX Master mouse that it is still very popular in the market.

In order to listen the messages exchanged in the pairing process, we utilized Ubertooth [10], a open source software, hardware and firmware to sniff and analyse Bluetooth and BLE packets.

In order to get the long term key exchange between the BLE devices, we used Crackle [17], an open source software which can take the packets derived from Ubertooth and obtain the keys generated in the connection process.

After decrypting the packets with Crackle, we could visualized the entire LE Legacy Pairing protocol using Wireshark.

7 Evaluation

7.1 Active Eavesdropping Attack

When improving the MITM attack described above, not being able to intercept the sound can be addressed by using a lower level control, which would be beyond the scope of the project. We've shown that discrete Man In the Middle attacks on headphones using `JustWorks` are plausible with unsophisticated hardware. Given the nature of uses of headsets and headphones, such as personal use and communication, the design should be deemed insecure.

7.2 Passive Eavesdropping Attack

We were able to extract the LTK and STK in a BLE connection created with the BLE 4.1 Logitech MX Master mouse. The demo of this attack can be found in this link.

Once we were able to sniff all the necessary messages to proceed with our attack, we obtained TK, STK and LTK almost instantly. Since there are not that many possible values for TK, Crackle is able to obtain this value in around a second [9].

However, this attack was not easy to deploy. We needed several attempts in order to get all the necessary messages since the Ubertooth sniffer is not able to capture all BLE packets at all times. In addition, this attack is not very practical, since the only vulnerability comes in the process of generating the LTK. Once this key is computed, the communication is secure under passive eavesdropping attacks.

Despite these technical difficulties, we still recommend to switch, if possible, to Bleutooth devices that use SSP or BLE devices with version 4.2 and beyond. If the right conditions are given, a passive eavesdropper could analyze all the messages offline and get the exact content sent by a particular BLE deviced, as shown by our experiment. This is not possible under SSP or LE Secure Connections, which are proven to be secure under passive eavesdropping attacks.

8 Suggestions for Future Work

We can expand the Active eavesdropping attack by applying to more devices using `JustWorks` , such as keyboards and BT mouse. Given the input nature of the devices, intercepting information from either of those can reveal a lot of information - from navigation on the computer to typing using on screen keyboard to intercepting keystrokes and password.

On the other hand, it's relatively easy to address these issues. In the case of headphones, a set of beep patterns can be used to verify the authenticity of the connection, given a display (to display a pattern) or an input (to 'tap' the pattern). Similarly, in the case of BT mouse, we can use click patterns, and BT keyboards should enforce typing PIN codes.

The next steps for the passive eavesdropping attack is to use the cracked LTK to decrypt the mouse packets and analyzing the data that it sends. In addition, we can expand this attack to other sensitive devices that utilize BLE such as keyboards and medical devices. We can use this data to simulate attacks that reveals sensitive information such as passwords inputted by clicking, bank account information type by users in a keyboard and important health related data outputted by medical devices.

In addition, we could combine both our active eavesdropping and passive eavesdropping to inject and modify packets maliciously to interfere normal communication between the devices.

Moreover, we could do more extensive research about the theoretical and practical vulnerabilities introduced in the most recent Bluetooth version 5.0.

9 Conclusion

Due to the widespread use of Bluetooth, the system must enforce security principles such as authentication, confidentiality, authorization and integrity. Passwords, phone calls and sensitive financial or medical data can be transmitted through Bluetooth devices, and users are often not aware of the risk that this imposes. As we demonstrate in this report, several arbitrary Bluetooth devices are still susceptible to both active and passive eavesdropping attacks. Bluetooth devices with the JustWorks association model are vulnerable under MITM attacks. Older BLE devices with version 4.0 and 4.1 are vulnerable to passive eavesdropping attacks. Despite the existence of newer versions of Bluetooth, there are still billions of devices with older versions still in use that pose a significant security vulnerability.

Code

All code relevant to our experiments and this project can be found in the following repository:

`https://github.mit.edu/samsonov/6857FinalProj`.

References

[1] *Bluetooth Core Specification v5.0*, Bluetooth Special Interest Group (SIG), Dec. 2016.

[2] *Bluetooth Core Specification v4.1*, Bluetooth Special Interest Group (SIG), Dec. 2013.

[3] S. (n.d.)., *Bluetooth low energy (BLE) enabled devices market volume worldwide, from 2013 to 2020 (in million units)*, 2015. [Online]. Available: https://www.statista.com/statistics/750569/worldwide-bluetooth-low-energy-device-market-volume/

[4] J. Padgette, J. Bahr, M. Batra, M. Holtmann, R. Smithbey, L. Chen, and K. Scarfone, *Guide to Bluetooth Security: Recommendations of the National Institute of Standards and Technology (Special Publication 800-121 Revision 2)*. USA: CreateSpace Independent Publishing Platform, 2017.

[5] K. Haataja, K. Hyppönen, S. Pasanen, and P. Toivanen, *Bluetooth Security Attacks: Comparative Analysis, Attacks, and Countermeasures*, ser. SpringerBriefs in Computer Science. Springer Berlin Heidelberg, 2013. [Online]. Available: https://books.google.com/books?id=bFm6BAAAQBAJ

[6] D.-Z. Sun, Y. Mu, and W. Susilo, "Man-in-the-middle attacks on secure simple pairing in bluetooth standard v5.0 and its countermeasure," *Personal Ubiquitous Comput.*, vol. 22, no. 1, pp. 55–67, Feb. 2018. [Online]. Available: https://doi.org/10.1007/s00779-017-1081-6

[7] P. Cope, J. Campbell, and T. Hayajneh, "An investigation of bluetooth security vulnerabilities," in *2017 IEEE 7th Annual Computing and Communication Workshop and Conference (CCWC)*, Jan 2017, pp. 1–7.

[8] M. L. Das and R. Mukkamala, "Revisiting bluetooth security (short paper)," in *Information Systems Security*, R. Sekar and A. K. Pujari, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 132–139.

[9] M. Ryan, "Bluetooth: With low energy comes low security," in *Proceedings of the 7th USENIX Conference on Offensive Technologies*, ser. WOOT'13. Berkeley, CA, USA: USENIX Association, 2013, pp. 4–4. [Online]. Available: http://dl.acm.org/citation.cfm?id=2534748.2534754

[10] "Software, firmware and hardware designs for ubertooth." [Online]. Available: https://github.com/greatscottgadgets/ubertooth/

[11] *Samsung Galaxy S7 Edge specifications*, Samsung. [Online]. Available: http://www.samsung.com/global/galaxy/galaxy-s7/#!/spec

[12] M. Duflot, M. Kwiatkowska, G. Norman, and D. Parker, *A Formal Analysis of Bluetooth Device Discovery*. [Online]. Available: http://qav.comlab.ox.ac.uk/papers/sttt-bluetooth.pdf

[13] *Assigned numbers for Baseband identifies the Inquiry Access codes and Class of Device/Service (CoD) fields.*, Bluetooth. [Online]. Available: https://www.bluetooth.com/specifications/assigned-numbers/baseband

[14] "bluetoothd(8) - linux man page." [Online]. Available: https://linux.die.net/man/8/bluetoothd

[15] "Change your bluetooth device mac-address." [Online]. Available: http://blog.petrilopia.net/hacking/change-your-bluetooth-device-mac-address/

[16] K. Ren, "Bluetooth pairing part 3 low energy legacy pairing passkey entry," Bluetooth. [Online]. Available: https://blog.bluetooth.com/bluetooth-pairing-passkey-entry

[17] "Crack and decrypt ble encryption." [Online]. Available: https://github.com/mikeryan/crackle

[18] U. Rijah, S.Mosharani, S.Amuthapriya, M. Mufthas, M. Hezretov, and D. Dhammearatchi, "Bluetooth security analysis and solution," in *International Journal of Scientific and Research Publications*, ser. WOOT'13, vol. 6. IJSRP, 2016.