Carnegie Mellon University in Qatar

Bluetooth Final Report

Independent Study | School of Computer Science

Gulnaz Serikbay <gserikba@andrew.cmu.edu>

Supervised by: Ryan Riley, Carnegie Mellon University

April, 2022

Abstract

Bluetooth Low Energy (BLE) is one of the most used wireless communication protocols supported by nearly all modern operating systems. BLE has been an integral part of smart home, health sectors and personal wearable devices. With the increasing significance of Bluetooth-embedded systems, it is vital to identify potential security vulnerabilities in Bluetooth-connected devices. Since the data transferred over these connections might be sensitive, users are vulnerable to privacy attacks through their wearable devices, wireless mouse, smart locks, smartwatches, wireless headphones, etc.

This study attempts to analyze the security of BLE pairings, specifically focusing on the pairings for Human Interface devices like BLE-enabled mouse. The experimental setup exploits Bluetooth USB dongles and Ubertooth One, an open-source 2.4 GHz wireless development platform used for making Bluetooth experimentations.

Contents

1 INTRODUCTION

Bluetooth Low Energy (BLE) has become an important wireless technology for the IoT industry. BLE has the advantage of very low power consumption over Bluetooth Classic, which makes it suitable and energy-efficient for many applications. It's largely used by embedded sensors that benefit from the fast connection, maximized idle time, and bursty lower bandwidth data transmission [1]. BLE radio transmits data over 40 channels in the 2.4GHz frequency with 2MHz spacing and supports many communication topologies, which provides developers with the opportunity to build reliable, large-scale device networks [4]. It's notable that BLE versions are backward compatible, but incompatible with Bluetooth Classic due to differences in implementation.

Despite the security improvements made to newer versions of Bluetooth Smart, BLE connections are at risk of many known attacks, which include, but not limited to:

1. Bluejacking: allows to send unauthorized data from one BLE device to another

2. BLE Spoofing Attack (BLESA): allows to steal a trusted connection through bypassing key authentication during reconnection; targeted towards a specific device

3. Man-In-the-Middle (MITM): allows an attacker to insert himself in between an existing BLE connection, where both slave and master devices trustfully communicate with an attacker, pretending to be a master and a slave for the two endpoints. Attack scenario allows to manipulate and send new data packets in the process [4].

2 BACKGROUND

Bluetooth connections established between a master (initiator) and a slave device are usually called piconet. A slave device, also called the Peripheral, is the one which advertises and accepts connections, while a master, the Central, is the one to scan, connect to the Peripheral and start the encryption process (if the data is encrypted).

The security features of Bluetooth technology are defined by the Bluetooth protocol stack according to Bluetooth Core Specifications. Each layer in BLE protocol stack has its own security features, so understanding the protocol stack is significant part of examining Bluetooth security.

2.1 BLE Protocol Stack

BLE architecture consists of 3 main components as illustrated in Figure 1: the controller, the host, and the application. The controller has a Physical Layer that employs analog communication, Link Layer, responsible for connection establishment, and A Direct test Mode used for certification radio frequency tests [1]. Host Controller Interface is an intermediate interface, followed by L2CAP, Attribute Protocol (ATT), Generic Attribute Profile (GATT), SMP and Generic Access Profile (GAP).

GAP is responsible for the discovery of a BLE network, which defines 4 roles for BLE devices: Broadcaster, Observer, Peripheral and Central. Peripheral and Central are the roles of our interest, since Broadcaster and Observer refer to devices with no connection.

ATT is responsible for the exchange of all user data and profile information in a connection and defines a database of device-specific attributes [4]. The device with the attribute database is referred to as a server, while the other end of the connection is a client. Client accesses

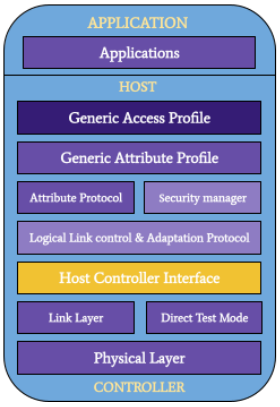

Figure 1: BLE protocol stack

the database of the server using ATT. For example, in a master-mouse pairing where mouse operates as a slave, master device accesses the attributes defined for the mouse. ATT defines the client/server protocol for data exchange. Data is transmitted in the format of attributes: handle, universally unique identifier, value, and permission. Handle is 16-bit uniquely assigned to a device for identification of a specific attribute on a particular GATT server, and is guaranteed not to change between transactions, and even across different connections with bonding. UUID defines the data type and permissions define the extent of access to data (read/write, encrypted read/write, authenticated read/write, authorized read/write) [4].

The link-layer defines 5 states for BLE devices: (1) Standby - default mode, (2) Advertising - peripherals allow to discover the device, (3) Scanning - device scans for advertising packets, (4) Initiating - device starts the connected request and (5) Connected - the device has established a connection. Connection is established when one of the devices initiates the communication with connect_req packet.

[Gulnaz: added info about channels here] Among the 40 channels that BLE uses, 3 channels (37, 38, 39) are dedicated for advertising and the rest 37 are for data communication. BLE devices that are in connected state exchange data through a channel determined by Adaptive Hopping pattern, which switches the carrier frequency using a pseudo-number generator defined by the host device. This brings challenges to sniff ongoing data transmissions. Devices in advertising state broadcast their advertising packets following a fixed hopping pattern, which switches channels in a predictive manner. Thus, targeting the connection is not easy if the sniffing device monitors a single channel.

BLE packet structure is the same for advertising and data packets with Preamble, Access Address, PDU and CRC shown on Figure 2. It's important to note that CRC is not a security feature for BLE since it's not cryptographically protected and only check for data integrity against noise [4].

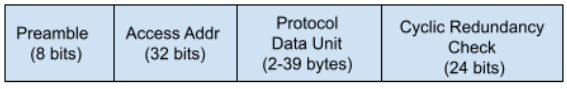

Figure 2: BLE packet structure

Bluetooth version	Key Features
Bluetooth 4.0, 4.1	Bluetooth Low Energy Legacy Pairing
Bluetooth 4.2	Low Power IP LE Privacy 1.2 LE Secure Connections
Bluetooth 5.0+	Slot Availability Mask (SAM) LE Long Range LE Advertising Extensions

Table 1: Bluetooth key features by versions

2.2 BLE Pairing

BLE devices can have public and private addresses discoverable for pairing. Private addresses protect the device from privacy vulnerabilities with a property of periodically changing addresses.

When devices form a connection, they can perform pairing, which can take different modes and thus different levels of security. If the pairing keys are reused in repeated pairings, the paired devices are called to be bonded.

Pairing consists of 3 phases described below:

First phase consists of devices sharing their capabilities to choose the pairing method in an unencrypted form (Pairing Feature exchange packet). The pairing methods determine how the Temporary Key is generated.

LE Legacy Pairing defines 3 ways of handling pairing while Secure Connections has Numeric Comparison mode in addition to them. Since Legacy Pairing is of our concern for this study, we can consider the following 3 association models of handling pairing [1]:

1. **JustWorks (JW):** Most common pairing mode, used in devices without a display or when the MITM flag in Feature Exchange packet is set to 0; TK-000000. This method just provides a way of establishing a connection, hence does not offer a protection. Used for small wearable devices, mice, headphones, etc.

2. **Passkey:** the pairing is verified using the six-digit passkey, which can be brute-forced. Passkey is used to create a Short Term Key, which is used to authorize the connection. It requires for a device to have a display or keyboards since the user must type the requested passkey manually. This protects the connection from the MITM attack.

3. **Out of band (OOB):** Rarely used, shares the pin using an out-of-band channel, where some portion of data packets are transferred over an alternative wireless protocol instead of BLE; TK requires 128 bits. The security of the pairing depends on the alternative protocol being used. E.g.: if a wearable device and the host device both support near-field communication (NFC), then pairing can be established by tapping 'Confirm' button from one/both of the devices via NFC protocol.

Devices exchange the pairing keys in the second phase, which determines the level of security of the connection. First, TK is sent in encrypted form and is used to derive other keys. If OOB or Passkey methods were used, both parties exchange 128-bit confirmation values (sConfirm,

mConfirm) derived from the TK, to authenticate the pairing. Devices then share their random values, so that the other endpoint could calculate their confirm values and if the confirm values match from both ends, they authenticate each other since they agreed on the same TK. Then, TK and Random values from slave and master(sRand, mRand) are used to calculate the Short Term Key (STK) to encrypt the link layer data.

Third phase of pairing is an optional phase, where devices exchange another set of keys to verify the pairing for future communication. Since devices are paired, they can transmit data, but if the first phase agreement requires the LTK, STK is used to generate LTK, which is then stored for future use.

Security features for Bluetooth Low Energy differ from those for Bluetooth Basic Rate/Enhanced Data Rate (BR/EDR) protocol, known as Bluetooth Classic. It uses 'key transport' instead of 'key agreement', where the link key is determined by both devices. Bluetooth 4.0 and 4.1 devices use LE Legacy Pairing, which exploits a custom key exchange protocol unique to the BLE standard. The security of the pairing process is dependent on the TK exchange method.

LE Secure Connections is a pairing method introduced with Bluetooth 4.2 as seen from Table 1. The security and privacy of 4.2 connections are enhanced and compatible with Bluetooth Classic. For example, if the two devices support BR/EDR and Bluetooth Smart, and both support Secure Connections on both sides, the pairing has to be done once. The LTK generated during the LE pairing process can be converted to a BR/EDR link key and a BR/EDR link key during the SSP pairing process can be converted to an LTK.

LE Secure Connections makes use of Diffie-Hellman public key exchange protocol, and authentication stage is separated into 2 parts (different for each pairing method), and authentication keys (sRand, mRand) are not utilized to generate the STK, LTK. Notably, only public keys are exchanged during pairing only to verify the authenticity, and private keys are used to generate keys[4]. To this end, it offers a clear much secure connection. Figure 3 outlines the pairing procedure for LE Secure Connections.

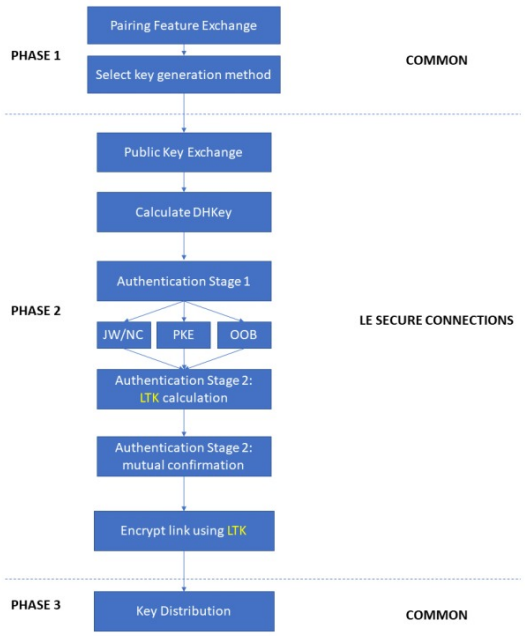

Figure 3: LE Secure Connections pairing procedure [4]

3 METHODOLOGY

3.1 Sniffing BLE

There are several available Bluetooth sniffers, including but not limited to Ubertooth One, Bluefruit, ESP32's BLE hardware, Sena UD-100, small devices like SMK-Link Nano and Bluetooth USB dongles. For this study, we exploit Ubertooth One as a sniffing device as it is programmable and the software utilities are open-source. Ubertooth software includes ubertooth-btle, a BLE utility for Ubertooth that supports following and promiscuous modes. It can produce Packet Capture (PCAP-formatted) files that contain a live network packet data logged in a certain format. PCAP logs are then are used to view and analyze the BLE communications.

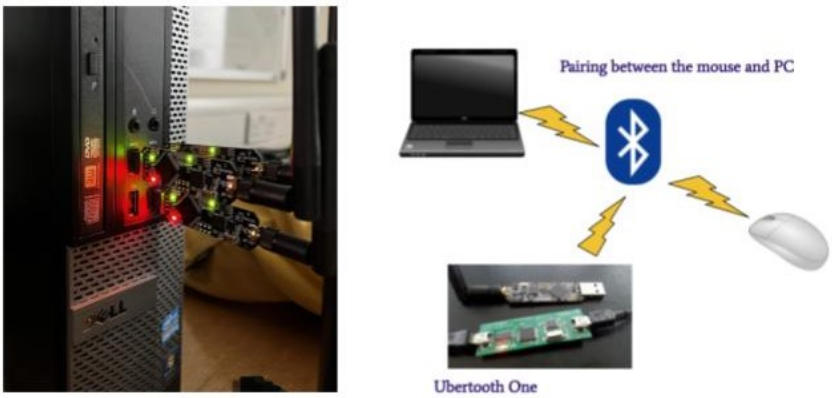

Figure 4: Experiment setup for sniffing packets with Ubertooth One illustrated

Wireshark [3]. is an open-source software suitable for analyzing network protocols that runs on Windows, Linux, macOS, etc. It has a set of powerful features of live capturing, display filtering, inspection of many protocols, not limited to Bluetooth and decryption support for some popular network protocols.

After installing needed software and packages, sniffing BLE traffic with Ubertooth One can be directly captured using Wireshark by creating pipes. It will capture a lot of unfiltered, potentially malformed BLE packets, which can be refined using Wireshark filters (see Appendix for examples). It is useful to take a look at the I/O graph provided by Wireshark of the sniffed packets to visualize the rate of BLE communications captured around and check if the targeted communication is indeed being captured. With a single Ubertooth One, the rate of capturing all the connection packets is around 1/3, since there are 3 advertisement channels. For complete capture of packets, 3 Ubertooth One devices were set up, each following one channel (37, 38, 39) as shown in Figure 4.

3.2 Decryption of pairing packets

Capturing the complete pairing packets with Ubertooth One allows to inspect pairing packets and obtain the TK, LTK values that are then used to decode the data packets between the two endpoints using crackle.

crackle [5] is a tool designed by Mike Ryan that exploits a flaw in the BLE pairing process, allowing a passive eavesdropper to decrypt data sent over BLE connection. Crackle brute forces the TK and uses the pairing packets collected using a BLE sniffer (Ubertooth One) to obtain the

Short Term Key (STK) and Long Term Key (LTK). The LTK can then be used to decrypt all the future communication between the master and slave devices. It works in 2 modes: Crack TK and Decrypt with LTK. Crack TK mode exploits the fact that the TK in Just Works and 6-digit PIN is a value in the range [0, 999999] padded to 128 bits.

BLE devices were examined for compatibility for experiments since crackle doesn't support pairings of LE Secure Connections introduced with Bluetooth 4.2+ versions. Figure 5 shows some failed attempts to decode LE traffic with crackle.

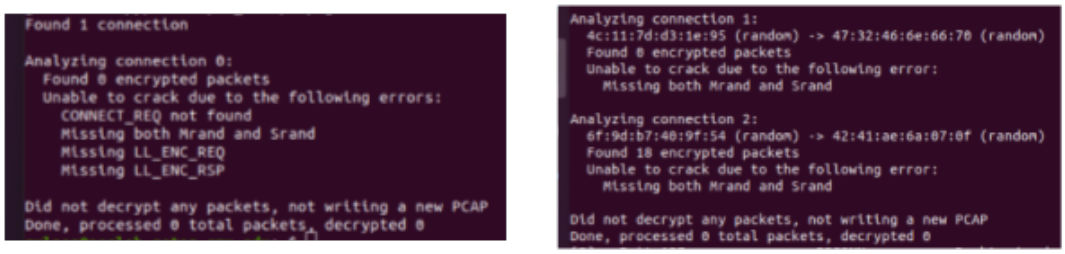

Figure 5: Unsuccessful attempts with crackle

As shown from the Figure 6, decrypted packets can be displayed and examined in detail with Wireshark's interface. The figure provides an outline of pairing packets (discussed in BACKGROUND) starting with connection request (CONNECT_REQ) initiated towards a slave with Bluetooth Address 36:10:0e:11:3d:0e (mouse's address) and Phase 1 feature exchange packets (LL_FEATURE_REQ, LL_FEATURE_RSP). Column 'Protocol' defines which layer is responsible for the packet being displayed. Then Pairing is authorized (Phase 2) using Confirmation and Random Values from both ends - slave and host. We can also notice that bonding was requested in the first phase, so the pairing is using the LTK for encrypting link layer packets for all future pairings. LL_ENC_REQ, LL_ENC_RSP packets indicate that transmitted data is being encrypted (using LTK), and Read By Value requests, responses indicate that BLE notifications are being sent from the server and received by the host.

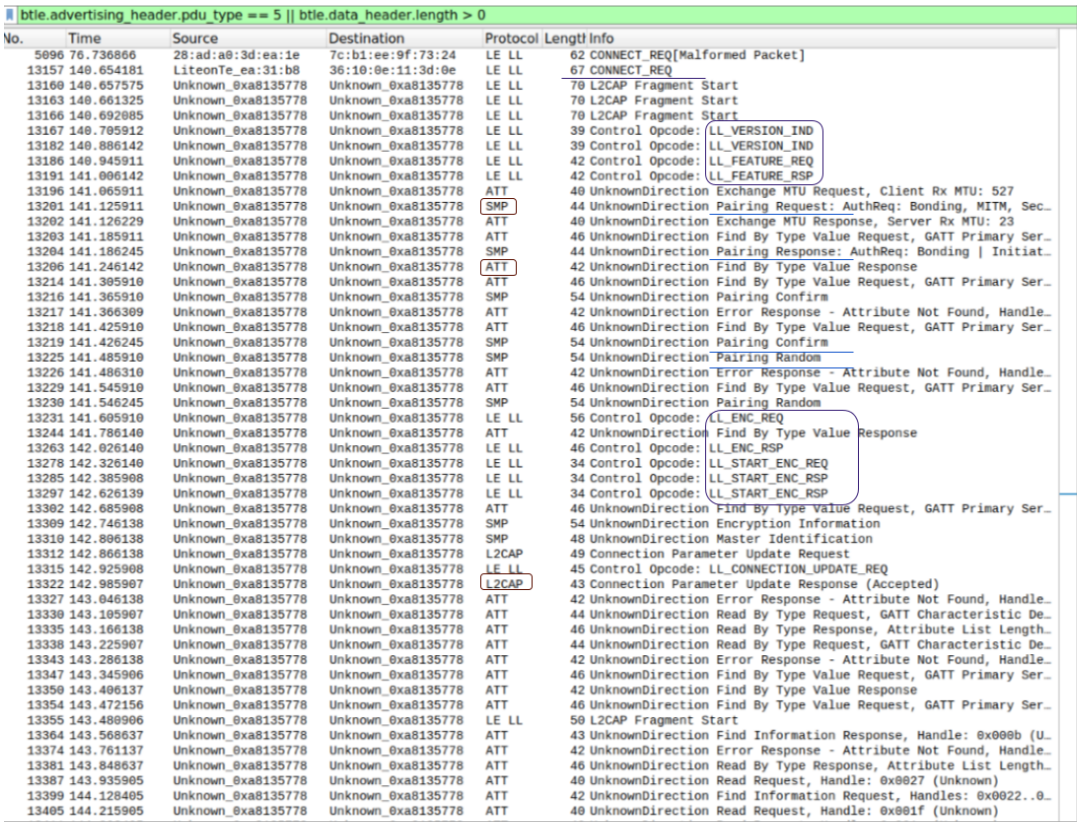

Figure 6: BLE pairing captured with Wireshark

3.3 Performing MITM

A systematic way of performing simple BLE attacks was found to be convenient with another open source framework called Mirage [2]. It's designed for security analysis experimentation with wireless technologies. Launched by Romain Cayre, this framework has support for Ubertooth, HCI devices, BTLEJack and other hardware tools. Mirage offers a Command Line interface and has powerful capabilities to perform security experimentation and build upon existing modules.

For the purpose of our study, several mirage modules such as ble_master, ble_mitm, ble_slave were used with 2 HCI devices (Bluetooth USB dongles) to spoof the mouse address and simulate a slave device through performing a simple MITM attack.

The Generic Attribute Profile (GATT) layer defines device's capabilities and services in a table of attributes ordered by unique identifiers, handles. It simply consists of attributes like GATT services, characteristics and descriptors that allow the mouse (any device in general) to be discoverable as a specific instance of mouse. The GATT layer data for the mouse was obtained through running the command 'discover' provided by mirage's ble_master module after connecting the simulation of the master device with HCI device plugged into laptop.

Then, HCI device can be used to simulate a slave device using this GATT layer information and connected to a master device using the BD_ADDRESS of the slave device. In our case, mouse's Bluetooth address, advertising and scan response packets were used to make the HCI device to be discoverable for master devices as a mouse.

Device	BT version	Pairing mode	cracked LTK?
WLDuoMouse	4.0	Legacy Pairing	Yes
Logitech Mouse	4.0	Legacy Pairing	Yes
Logitech Keyboard	4.2	Secure Connections	No
QY8 Headset	4.1	Legacy Pairing	No

Table 2: Results of cracking the LTK for several devices

4 RESULTS

It was possible to sniff livestream packets with Ubertooth One from discoverable devices and display them over Wireshark's interface. Using Ubertooth One, BLE sniffing experiments were targeted for several master-slave combinations like mouse-laptop, mouse-phone, keyboard-laptop, headset-phone, etc. Exploiting key-brute forcing capability of crackle, it was achieved to obtain Temporary Key, Short term Key and Long Term Key for the pairings with mouses. Since crackle's key-cracking method does not support all kinds of pairing modes and Bluetooth versions, the scope of success for cracking the LTK was limited. Table 2 presents the results for devices that were tested. Both cracked pairings were in JustWorks mode, where the TK is just 000000.

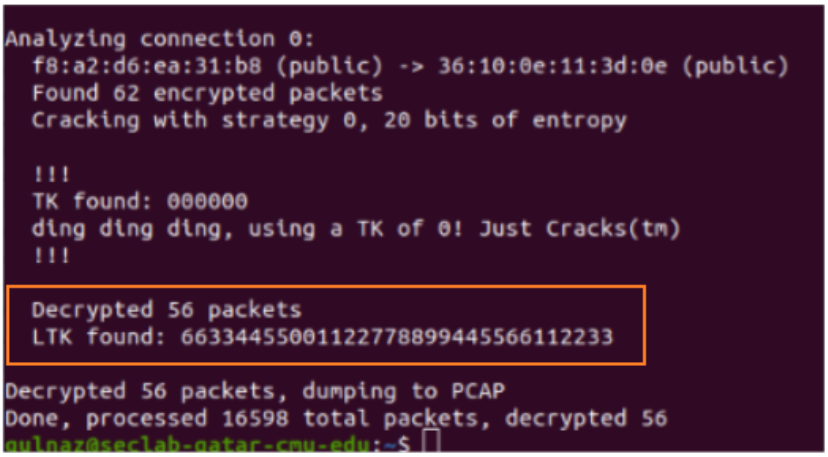

Figure 7: LTK obtained with crackle

More interestingly, for one of the mice (WLDuoMouse), the same LTK was generated when paired with different host devices. As seen from Figure 7, the LTK doesn't seem to be randomly generated and clearly hardcoded, which implies a major security issue.

When the server (mouse) sends data to the connected host device, they use a GATT procedure provided by BLE specification called characteristic value notification [4]. Using the pairing keys (TK, LTK) unique to connection between Logitech mouse and laptop, it was possible to decode the handle and values corresponding to different notifications, mouse clicks. This was possible because the BLE pairing had bonding, where pairing keys are reused and remain unchanged in future communications.

Mouse clicks are identified by a handle 0x19, and values are presented in Table 3. The mouse notification payloads (Protocol Data Unit in Figure 2) consist of 6 bytes, where each byte identifies certain mouse input. According to our inspections, the upper byte of the payload is responsible for left/right clicks, while the lower byte is responsible for the up/down scrolls. Mouse

Mouse command	Inspected Value
Mouse Left Click	010000000000
Mouse Right Click	020000000000
Mouse Middle Click	030000000000
Mouse Click released	000000000000
Mouse Scroll Up	000000000001
Mouse Scroll Down	0000000000ff
Mouse Move	00c8ffebff00

Table 3: Inspected notification values for mouse commands

coordinates are represented in middle 4 bytes, where 2-3rd and 4-5th bytes are responsible for relative x-coordinate, y-coordinate values, respectively Importantly, notification values for 2 different brands of mouse (Logitech, Platinum WLDuoMouse) followed the same format.

Using Mirage, it was possible to extract GATT layer data for the Platinum mouse (shown in Figure 8) that revealed some device-specific system design details. This allowed to simulate a slave device with HCI device by making it discoverable as a mouse for scanning devices like laptop and phone. Since the HCI device shares the same service and attributes as the mouse in addition to the same public address, any master device cannot differentiate between the mouse and the HCI device.

Attributes

Attribute Handle	Attribute Type	Attribute Value
0x0001	Primary Service	0018
0x0002	Characteristic Declaration	0a0300002a
0x0003	Device Name	WLMouseDuo
0x0004	Characteristic Declaration	020500012a
0x0005	Appearance	c203
0x0006	Characteristic Declaration	020700042a
0x0007	Peripheral Preferred Connection Parameters	0600060064002c01
0x0008	Primary Service	0118
0x0009	Characteristic Declaration	200a00052a
0x000a	Service Changed	
0x000b	Client Characteristic Configuration	
0x000c	Primary Service	0a18
0x000d	Characteristic Declaration	020e00292a
0x000e	Manufacturer Name String	YiChip
0x000f	Characteristic Declaration	021000502a
0x0010	PnP ID	02351222aa0100
0x0011	Primary Service	1218
0x0012	Characteristic Declaration	0613004e2a
0x0013	Protocol Mode	01
0x0014	Characteristic Declaration	1215004d2a
0x0015	Report	
0x0016	Client Characteristic Configuration	
0x0017	Report Reference	0201
0x0018	Characteristic Declaration	1a19004d2a
0x0019	Report	
0x001a	Client Characteristic Configuration	
0x001b	Report Reference	0101
0x001c	Characteristic Declaration	121d004d2a
0x001d	Report	
0x001e	Client Characteristic Configuration	
0x001f	Report Reference	0301
0x0020	Characteristic Declaration	0e21004d2a
0x0021	Report	
0x0022	Report Reference	0202
0x0023	Characteristic Declaration	5K*
0x0024	Report Map	05010902a10185010901a10005091901290815002501
0x0025	Characteristic Declaration	1a2600332a
0x0026	Boot Mouse Input Report	
0x0027	Client Characteristic Configuration	0100
0x0028	Characteristic Declaration	0229004a2a
0x0029	HID Information	
0x002a	Characteristic Declaration	042b004c2a
0x002b	HID Control Point	

Figure 8: A table of GATT layer data for the mouse

It is also possible to crack the LTK and pairing configurations using mirage's ble_mitm module, which shows the process of determining sRand, mRand, STK and LTK more interactively to the user side. Obtaining the LTK then allows to track the signals sent by both master and slave devices. Decoded packets can also be monitored more interactively as seen from Figure 9.

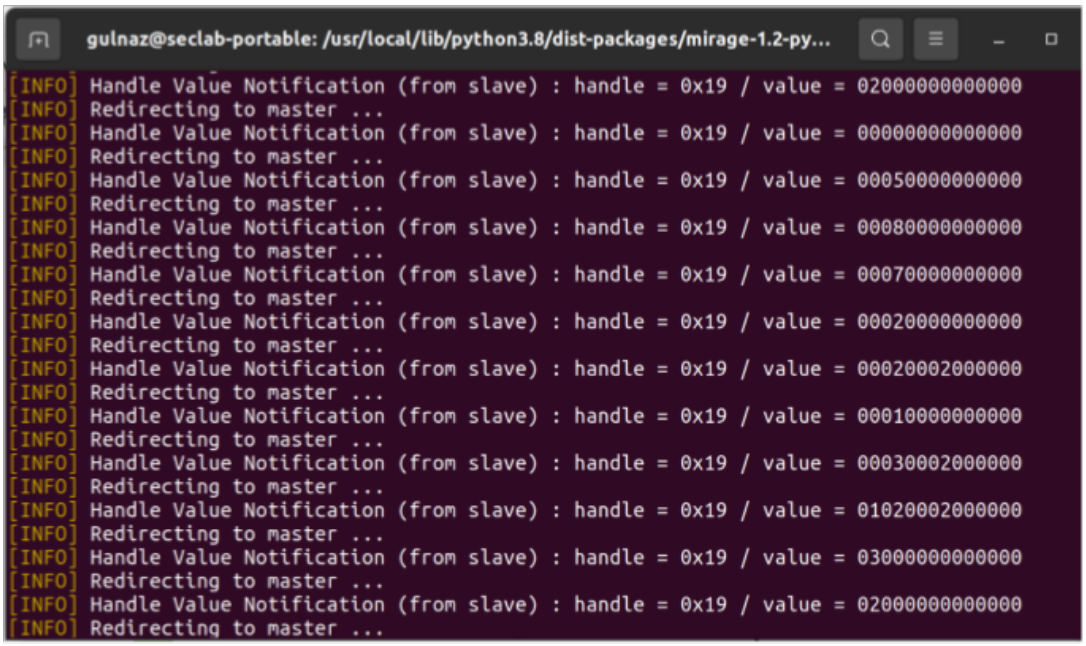

Figure 9: Tracking packets with mirage

Over several traces, the Bluetooth device address (BD_ADDRESS) of the WLDuoMouse was changing, which made the process of performing the targeted MITM inconvenient with mirage. This shows that the manufacturer decided to protect the device from being tracked with its public address.

However, from several traces, it was noticeable how the BD_ADDRESS of WLDuoMouse followed a predictable increasing pattern. Even though the discoverable address changed with every pairing, for short time tracking, it can be easily guessed by brute forcing the last bytes. Hence, it was concluded that this protection can be overcome.

BD_ADDRESSES for a series of experimentations:
36:10:0E:11:7A:12
36:10:0E:11:7B:BD
36:10:0E:11:7C:5B
36:10:0E:11:7D:16
36:10:0E:11:7E:A4
36:10:0E:11:7F:9A
36:10:0E:11:81:EB
36:10:0E:11:85:CC

Using the decoded values (Table 3) for different mouse commands sent from the server device (slave), the communication between the paired devices can potentially be manipulated.

A potential direction for developers studying the security of BLE would be to implement modules and scenarios for open source projects like Mirage and Ubertooth. This study has shown that the existing communication for a master-slave pairing can be hijacked, but it is also possible to create independent connections with a fake slave device (HCI). Specifically, one can create a device class with Python script for the mouse using all the available GATT layer data, Advertising and SCAN_RSP packets. Then this script can be run on HCI device with the spoofed address and simulate a slave device that can send a series of programmed clicks to the master

device. An example of such possibility was created for a keyboard, and can be viewed on this link.

Bluetooth LE can be considered as one of more reliable and secure wireless technologies, and this is supported by the results of this study. The differences in Bluetooth protocol implementations, pairing methods, and device-specific communication configurations make the process of hijacking the BLE connection hardly generalizable.

5 CHALLENGES

Future research on BLE should be directed towards the development of platforms that allow for laboratory experiments with BLE connections. Many of the currently available tools supporting Ubertooth One and Adafruit sniffers had limited range of functionalities and poor compatibility with modern versions of programming libraries, BLE devices and pairing modes. For instance, experimentation with Adafruit's Bluefruit device, some functionalities were not working properly due to Python environment incompatibility issues, which could not been resolved. Moreover, there is almost no recently published research utilizing this tool for Bluetooth sniffing purposes.To this end, it was concluded that some tools used with the Bluefruit were outdated.

Also, the scope of the experimentations was limited due to capabilities of some utilized software such as crackle. Since crackle expects the whole pairing to be captured in the input pcap file and accepts only Legacy Pairing, it was not possible for all pairings to obtain the LTK and inspect the values defined by ATT.

One possible reason for why sniffing some BLE pairings were unsuccessful is that the SMP Pairing Request and Pairing Response PDUs contain fields that define some security features: IO Capability, SC, MITM, Maximum Encryption Key Size, Initiator Key Distribution, and Responder Key Distribution. If MITM flag is set to 1, then authentication may be requested from the devices and not allow other devices from accessing the connection even when using Legacy Pairing[4].

6 RECOMMENDATIONS

It is recommended that the prospective research is narrowed down to a set of devices that share similar networking configurations. Firstly, it's better to focus the attention to devices with the same Bluetooth class capabilities. For example, Human-Interface devices (HID) are one instance of such classes, and should be explored distinctly from headsets or music players. For instance, some devices with LE Audio rely on time-synchronized connections and use synchronous channels unlike keyboards. Even though keyboards and mice both belong to HID class, pcap traces showed that they follow different key generation strategies for pairing. In general, Bluetooth keyboards have better security than mice. Most of them rely on Passkey pairing mode rather than JustWorks, which is known to be more secure.

Secondly, it is to focus on a set of devices from the same manufacturer. Bluetooth Core Specification offers many different security options for manufacturers to employ for each protocol stack layers. To this end, the implementation of Bluetooth pairings, data packets and Host-specific features are dependant on device manufacturer's design choices. This explains why we were not observing similar patterns when capturing data packets and experimenting with pairings for several devices. BLE-embedded devices have different combinations of pairing

modes, LTK and BD_ADDRESS creation strategies, etc.

Bluetooth security is dependent mostly on the security of Host and Controller components, which is defined by the device manufacturers. Hence, mitigation of security vulnerabilities are mainly maintained by BLE developers. It was concluded that the strategy used for creation of addresses plays an important role in limiting a potential attacker's actions. Most of the attacking scenarios, including the MITM scenario used in this study, rely on targeting the device by its BD_ADDRESS. Public addresses make the connections vulnerable for such targeting. Using Non-Resolvable Random Addresses is known to be the most secure way to prevent the device from being tracked.

This study did not analyse the security of the Application Layer in the BLE protocol stack, and majority of the current research studies focus on the Host-Controller components. It is important to identify security issues and study mitigation strategies to be employed in the Application layer.

7 CONCLUSION

This research was aimed at studying Bluetooth Low Energy and investigating vulnerabilities of some IoT devices. Overall, Bluetooth has proven to be more secure than some other wireless communication technology like Wi-Fi. Using Ubertooth One, Wireshark and crackle, we were able to perform spectrum analysis, packet-sniffing and decryption, thus allowing us to inspect the values being sent from the guest device to the host device and identify what information is being transmitted. Using BLE modules provided by mirage, we were able to extract GATT layer data for one of the devices. Using HCI devices, it was possible to spoof the address of the mouse, perform MITM sniffing and modify the data being transferred over the communication. Hence, we showed that Human Interface devices are vulnerable to simple attacks like MITM and suggested recommendations to create scenarios for spoofing attacks. Finally, we suggested some mitigation strategies to corresponding security issues.

8 APPENDIX

Launching Ubertooth and Wireshark:
Create a pipe with command:

```
$ mkfifo /tmp/pipe
```

Launching 2 Ubertooth One devices to monitor all 3 advertisement channels, setting each device to one of the channels:

```
$ mkfifo /tmp/fifopipe0 && mkfifo /tmp/fifopipe1 && mkfifo /tmp/fifopipe2
$ ubertooth-btle -U0 -A37 -f -c /tmp/fifopipe0 &
$ ubertooth-btle -U1 -A38 -f -c /tmp/fifopipe1 &
$ ubertooth-btle -U2 -A39 -f -c /tmp/fifopipe2 &
```

Wireshark can be launched from command line or via the interface:

```
$ wireshark -k -i /tmp/fifopipe0 -i /tmp/fifopipe1 -i /tmp/fifopipe2 &
```

Useful Wireshark filtering options:
1. filters out empty BLE packets
2. filters out malformed packets
3. shows only pairing related packets *CONNECT_REQ*, *L2CAP*, etc):

```
1. !(btle.data_header.length == 0)
2. !_wc.malformed
3. btle.data_header.pdu_type == 5
```

crackle commands:
For Crack TK, the command line expects the input as follows:

```
$ crackle -i input.pcap -o decrypted.pcap
```

Decrypt with LTK mode requires the user to enter LTK to decrypt the packets as follows:

```
$ crackle -i encrypted.pcap -o decrypted.pcap -l LTK
```

mirage commands:
Simulating the slave device can be launched using the command below, where FF:FF:FF:FF:FF:FF should be replaced by a target address and GATT_FILE should be given a filepath to load the GATT layer data:

```
$ sudo mirage "ble_scan|ble_connect|ble_discover|ble_adv|ble_slave"
ble_scan1.INTERFACE=hci0 ble_scan1.TARGET=FF:FF:FF:FF:FF:FF ble_scan1.TIME=5
ble_connect2.INTERFACE=hci0 ble_discover3.GATT_FILE=filename
```

Add GATT_FILE = "filepath" if the command doesn't automatically load the GATT layer data:

```
$sudo mirage ble_slave INTERFACE=hci1 GATT_FILE="/tmp/att.cfg"
```

References

[1] Daniela Catanzaro. “Study and investigation of Bluetooth Low Energy security in the IoT environment”. In: *Politechnico Di Torino* (2020).

[2] Romain Cayre. “mirage [Computer Software]”. In: (2019).

[3] Gerald Combs. “Wireshark [Computer Software]”. In: (1998). URL: https://www.wireshark.org/docs/.

[4] “Developer Study Guide: Bluetooth® Low Energy Security 1.1.3”. In: (2021). URL: https://www.bluetooth.com/bluetooth-resources/le-security-study-guide/.

[5] Mike Ryan. “crackle [Developer tool]”. In: (2013). URL: http://lacklustre.net/projects/crackle/.