Bluetooth Low Energy Makes ”Just Works” Not Work

Karim Lounis, Mohammad Zulkernine

Bluetooth Low Energy Makes "Just Works" Not Work

Karim Lounis and Mohammad Zulkernine
Queen's Reliable Software Technology Lab, School of Computing, Queen's University
Kingston, ON, Canada
{lounis,mzulker}@cs.queensu.ca

***Abstract*—BLE (Bluetooth Low Energy) is being heavily deployed in many devices and IoT (Internet of Things) smart applications of various fields, such as medical, home automation, transportation and agriculture. It has transformed the classic Bluetooth into a technology that can be embedded into resource constrained devices running on a cell coin battery for months or years. Most BLE devices that are sold in the market use the Just Works pairing mode to establish a connection with peer devices. This mode is so lightweight that it leaves the implementation of security to application developers and device manufacturers. Unfortunately, as the market does not want to pay for security, a number of vulnerable smart devices are strolling around in the market. In this paper, we discuss how Bluetooth devices that use the Just Works pairing mode can be exploited to become nonoperational. We conduct a case study on three different Bluetooth smart devices. We show how these devices can be attacked and abused to not work properly. We also present a vulnerability that is due to the behavior of BLE smart devices and the Just Works pairing mode. This vulnerability can be exploited to generate an attack that affects BLE availability. We propose a solution to mitigate the attack.**

***Index Terms*—Bluetooth Low Energy, Internet of Things, Just Works pairing, BLE security, Ubertooth, and Bluetooth attacks.**

I. Introduction

For the past five years, BLE (Bluetooth Low Energy) has become more and more popular and adopted in many IoT (Internet of Things) devices in a variety of fields. Nowadays, it can be found in high-end smartphones, sport devices, home appliances, vehicles, and medical devices. It has transformed the classic Bluetooth technology to embed into resource constrained devices that run on a cell coin battery for months or years. These devices adopt lightweight authentication and encryption protocols to establish secure connections. For example, to connect a Bluetooth smart device, such as a blood sugar monitor, to a smartphone in order to get a notification when the blood sugar level is up so that insulin can be taken, the smartphone and the blood sugar monitor must be paired, i.e., connected and authenticated. Nevertheless, most, if not all, Bluetooth smart devices that are available in the market, use a pairing scheme called Just Works. This scheme allows any device to connect to another device without any "secret-based" authentication. This makes these smart Bluetooth devices vulnerable to a variety of attacks, which threaten the security, privacy, and safety of Bluetooth users.

While developing the BLE specification, the Bluetooth Special Interest Group[1], or Bluetooth SIG for short, has focused more on power consumption than security. Although, the provided security is not that perfect, Bluetooth SIG has provided security recommendations and guidelines in their standard specification documents. These recommendations should be followed by application developers and Bluetooth smart device manufacturers to implement security into their devices. Notwithstanding, many device manufacturers and Bluetooth application developers do not follow these recommendations. This is mainly because vendors and application developers choose cheaper and faster option over better quality when it comes to "Quality, time, and cost, pick any two". This has resulted in millions of vulnerable smart devices strolling around in the market. Such devices can be exploited to generate various attacks that threaten the security, privacy, and safety of Bluetooth users.

In this paper, we present a vulnerability in the BLE Just Works pairing mode. Then through a practical case study, we evaluate the security of three Bluetooth smart devices: a bikelock, a lightbulb, and a deadbolt. These devices are made from different manufacturers and implement security in a different manner. We show how the Just Works pairing mode makes those devices vulnerable to different types of attacks. We propose a solution to mitigate the identified vulnerability as well, which will improve BLE security.

The remainder of the paper is organized as follows: Section II presents BLE technology. In Section III, we discuss the related work. We describe how attacks are generated on BLE devices in Section IV. In Section V, we present a vulnerability in the BLE Just Works pairing mode and explain how it can be exploited to abuse Bluetooth availability. Section VI evaluates the security of three Bluetooth smart devices and provides a solution to fix the identified vulnerability. Finally, Section VII concludes the paper.

II. Bluetooth Low Energy

In this section, we first give an overview about BLE (Bluetooth Low Energy) and then we discuss BLE security.

[1]Bluetooth SIG is the Bluetooth standard organization. It has hundreds of members, such as Ericsson, Nokia, Apple, IBM, and Microsoft.

978-1-7281-3949-4/19/$31.00 ©2019 IEEE

A. BLE Overview

Bluetooth is a wireless communication technology based on the IEEE 802.15.1 standard. It is used for exchanging data between fixed and mobile wireless devices within a short range and building WPANs (Wireless Personal Area Networks). It uses the free unlicensed ISM (Industrial, Scientific, and Medical) radio band at 2.4GHz and adopts the FHSS (Frequency Hopping Spread Spectrum) transmission technique to send packets while reducing interference. Bluetooth has evolved for the last twenty years, from Bluetooth v1.0 (1999) to Bluetooth 5 (2017), coming out with better power consumption, stronger security, higher data rate, and longer range.

In 2010, the Bluetooth SIG (Special Interest Group) released Bluetooth v4.0+LE (Low Energy), or simply BLE [1]. This new technology includes two sub-specifications: Bluetooth smart, also known as BLE single mode; and the Bluetooth smart ready, also known as BLE dual mode. These two sub-specifications have a completely different physical and link layers. Thus, there exist two different protocol stacks: the BLE dual mode and the BLE single mode as illustrated in Fig. 1. BLE dual mode implements both protocol stacks, the classic Bluetooth stack, which is used in Bluetooth v1.1 to Bluetooth v3.0+HS (High Speed); and the Bluetooth smart stack. Bluetooth devices which implement only the Bluetooth smart stack, i.e., single mode, are not retro-compatible with classic Bluetooth devices [1].

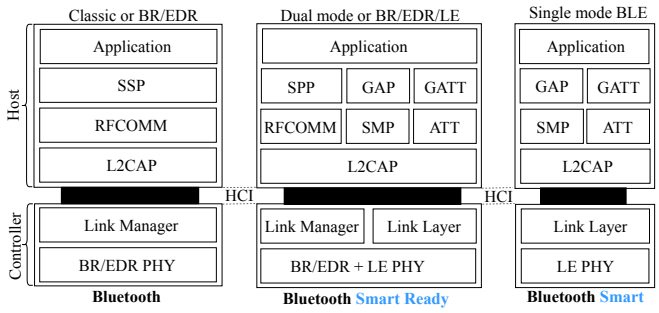

Fig. 1. Classic Bluetooth stack (leftmost), Bluetooth smart ready stack (center), and Bluetooth smart stack (rightmost).

Bluetooth low energy divides the spectrum into forty channels each of 2MHz bandwidth. Three channels out of the forty (37, 38, and 39) are used for advertisement and connection establishment. The remaining channels are used for data exchange during a communication. BLE stack can be divided into three layers: controller, host, and application. The controller layer deals with transmission and reception of radio signals. The host layer encompasses all the modules needed to provide applications with diverse APIs (Application Programming Interfaces) to communicate with remote Bluetooth devices through the radio. Finally, the application layer groups all Bluetooth smart applications. The host layer is composed of multiple modules. The L2CAP (Logical Link Control and Adaptation Protocol) is used for protocol multiplexing, packets segmentation and reassembly, quality of service, and group abstractions. The SM (Security Manager) defines all cryptographic functions needed to provide security services. The GATT (Generic Attribute Profile) defines a set of standard profiles that specify how messages are constructed, formatted, and exchanged between two Bluetooth devices. The GAP (Generic Access Profile) defines four BLE roles in which a device can operate: observer (can only receive traffic), broadcaster (can only advertise), peripheral (can accept connections from other devices), and the central (can initiate connections with peripherals) [1]. The peripheral and the central are classically known by slave and master, respectively.

The set of profiles that are defined by the GATT module are used to develop smart applications. Each profile defines a set of services. Each service is identified by a unique identifier UUID (Universal Unique Identifier). A service includes one or more characteristics. Each characteristic is identified by a value known as handle. A characteristic contains one or more properties, one value, and one or more descriptions. For example, a smart application running on a central device, such as a smartphone, can use the *Battery service* to monitor the level of the battery in a remote Bluetooth peripheral device.

B. BLE Security

In order to communicate, Bluetooth devices have to be connected and authenticated to each other. This is performed during an authentication procedure called pairing. The pairing procedure allows two Bluetooth devices to authenticate each other and negotiate on a set of security parameters to derive a master key called link key, or LTK (Long Term Key) in BLE. This key is stored and used to encrypt all future communications between a pair of Bluetooth devices. BLE defines two main pairing modes: legacy pairing and SC (Secure Connections). The legacy pairing applies the SSP (Secure Simple Pairing), which is used in classic Bluetooth (from Bluetooth v2.1+EDR to v4.1+LE), but without ECDH (Elliptic curve Diffie-Hellman) [2]. The Secure Connections upgrades SSP since Bluetooth v4.2+LE (and Bluetooth 5). It uses ECDH in BLE, longer keys, and provides data integrity. In BLE legacy pairing, only three association modes are possible: Just Works, Passkey Entry, and Out of Band. In BLE Secure Connections, a fourth mode called Numeric Comparison, is added. Besides Numeric Comparison, none of the previous association modes provides protection against passive eavesdroppers. BLE employs AES[2] 128-bit in CCM[3] mode for data encryption. In this paper, we only consider legacy pairing, i.e., Bluetooth v4.0+LE and v4.1+LE.

III. Related Work

Since BLE (Bluetooth Low Energy) was released, many research works have been conducted to study the security of BLE in general, and Bluetooth smart devices in particular. In [4], the authors have shown how BLE can be easily sniffed using affordable hardware, such as Ubertooth One [5]. In [6],

[2]AES (Advanced Encryption Standard), also known as Rijndael, is a symmetric cipher established by the U.S. National Institute of Standards and Technology (NIST) in 2001 [3].

[3]CCM: Counter with Cipher block chaining - Message authentication code.

the authors have demonstrated fundamental weaknesses in the key exchange protocol that is adopted in BLE. Ray *et al.* [7] and Lonzetta *et al.* [17] have discussed general attacks on BLE. With respect to Bluetooth smart devices, Ryan [6] have demonstrated attacks on heart rate monitors. Rose *et al.* [8] have discussed how they hacked twelve smart locks out of sixteen. Cauquil [22] have developed a MITM (Man In The Middle) framework, called Btlejuice, to conduct MITM attacks on smart padlocks, a robot, and a blood GMS (Glucose Monitoring System). Jasek [20] have demonstrated through different attacks, such as MITM attack, replay attacks, and reverse engineering of Bluetooth mobile applications, how they have managed to hack and unlock different types of smart padlocks. Tan [21] have presented how they have managed to hack bicycle locks used for bike sharing applications in Singapore. Zhang *et al.* [9] have conducted attacks on wearable devices, in particular, wristbands, and have shown how vulnerable these devices are. Similar work in [10], [11], have conducted attacks on the same types of devices, i.e., wristbands, by reverse-engineering the mobile applications that are used to connect and control the devices.

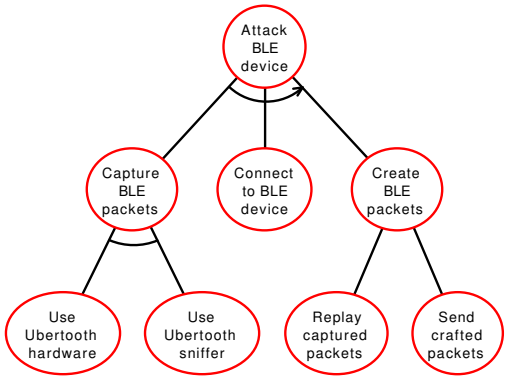

Fig. 2. An attack-tree for attacking Bluetooth smart devices, where ◯ represents an attack and ◯–◯ indicates an attack refinement. A refinement can be a conjunction (**And**), disjunction (**Or**), or a sequential conjunction (**Then**).

Many vulnerabilities as well as privacy issues have been discussed in [12], [13], where authors have shown that some smart devices, such as keyboards, fitness trackers, and heart rate monitors, do not employ encryption at all and all confidential and private information are sent in plaintext over the air. Gullberg [14] have presented denial of service attacks on BLE. In this paper, we conduct a case study on three different Bluetooth smart devices: a bikelock, a lightbulb, and a smart deadbolt, and show how vulnerable those devices are by generating attacks on different security services: confidentiality, authenticity, integrity, and availability. We also discuss a vulnerability that is due to the BLE Just Works pairing mode. This vulnerability can be exploited to deprive legitimate users from correctly using their smart devices.

IV. Attacking Bluetooth Low Energy

BLE (Bluetooth Low Energy) has been a hot subject of security attacks. Many vulnerabilities have been discovered and various attacks have been conducted and reported in the literature as well as in the hacking conferences. In the following paragraph, we discuss the attack anatomy that most attackers follow to generate attacks against BLE devices.

To conduct an attack on BLE devices, an attacker usually follows the attack steps described by the attack-tree[4] of Fig. 2, where ◯ can be a final goal, a sub-goal, or a basic attack, depending at which level the attack node ◯ is situated. ∧ refers to logical conjunction refinement (**And**) or sequential conjunction refinement if there is an arrow (**Then**), and ∨ refers to logical disjunction refinement (**Or**). The steps are as follows:

Step 1. The attacker starts by capturing BLE traffic, and more interestingly, capturing packets related to a target communication. Capturing a BLE communication requires the attacker to capture the *connect_request* packet that is sent during a connection establishment. This packet contains all synchronization parameters, such as the hop increment, the hop interval, timeout, window size, and the access address, needed to generate the hop sequence [1]. Knowing the hop sequence, an attacker can follow up a complete ongoing BLE communication. If the attacker misses this packet, it can apply other techniques, such as the one discussed in [6].

Step 2. Once the attacker has captured enough packets, it connects to the target device. As most Bluetooth smart devices employ the Just Works pairing mode, the attacker will easily succeed this step.

Step 3. Now that the attacker is connected, it can start replaying previously captured packets or generating new packets based on the captured ones. The attacker just needs to know which service (i.e., UUID) and characteristic (i.e., handle) it targets. As part of the BLE protocol, the attacker can request the smart device to reveal all services and characteristics that the device hosts. Once the attacker identifies the targeted service and characteristic, it generates read and write commands to perform unauthorized readings and writings on the smart Bluetooth device.

V. Just Works Vulnerability in BLE

In the following paragraphs, we discuss the features of the Just Works pairing mode in BLE (Bluetooth Low Energy) and how attackers can take advantage of these features to generate attacks. We present a vulnerability that can be exploited to abuse Bluetooth smart devices functionality. The Just Works pairing mode is a pairing procedure that is used when at least one of the Bluetooth devices to be paired with, does not have any input and output capability. In BLE, it is the most adopted pairing mode, as almost all Bluetooth smart devices, do not have any input and output capability such as lightbulbs, smart locks, fridges, keyfobs, heart rate monitors, and other devices. The main feature of this pairing mode is that it does not require any authentication to complete a

[4]Attack-trees [16] are graphical security models used to represent possible attack scenarios on a given information system in a user-friendly way.

pairing procedure. Technically, there is a kind of meaningless authentication, where both devices generate a shared STK (Short Term Key) by proving to each other the possession of a shared TK (Term Key), where this TK is by default set to `0x00` in this pairing mode. The STK is then used to exchange the LTK (Long Term Key) [1]. In few words, anybody can connect to any BLE device that uses Just Works. To provide security, Bluetooth application developers along with Bluetooth smart device vendors, use the BLE security specifications and follow the Bluetooth SIG recommendations to design and implement a proprietary high-level security protocol to secure their applications and Bluetooth smart devices. Thus, the authentication procedure of any application that is developed to connect and control a Bluetooth smart device, usually goes through two phases: (1) The standard Just Works phase. (2) The application-based authentication phase. The latter phase is not always implemented. Some vendors invest some money to implement the second phase and make their applications and BLE devices secure, whereas other vendors leave the pairing as is, i.e., not secure.

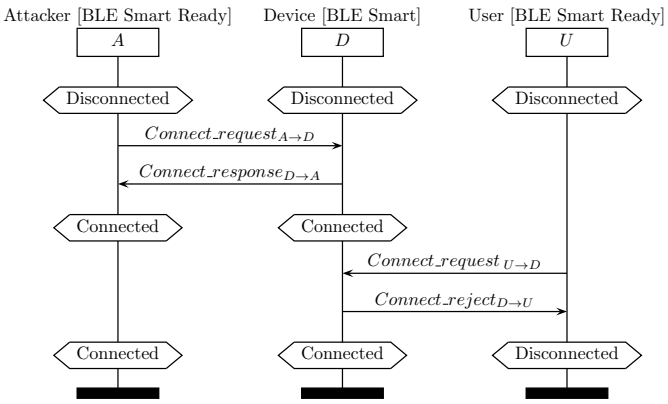

Fig. 3. Connection deprivation in BLE Just Works pairing. The notation $M_{x \to y}$ indicates a message sent from x to y, where $x, y \in \{A, D, U\}$.

Another feature in BLE is that most, if not all, Bluetooth smart devices, which operate as peripherals, are restricted to establish one and only one connection at a time with a central device. This can be interpreted as a security vulnerability that can affect Bluetooth availability. As anybody can connect to these devices, and as these devices accept only one connection at a time, an attacker can establish a connection with these devices and deprive legitimate users from connecting to them and using their services as illustrated in the MSC[5] of Fig. 3.

VI. Case study

To study the security of BLE devices and to demonstrate the impact of the vulnerability that we have presented in the previous section, we consider three use cases. The first use case consists of a Bluetooth bikelock, where the owner of a bike uses a mobile application installed on its smartphone to connect to the Bluetooth lock and unlock it. The second use case consists of a smart Bluetooth lightbulb, where the owner uses a mobile application to switch the lightbulb ON and OFF, and adjust its brightness. Finally, the third use case consists of a home Bluetooth deadbolt. The owner uses a mobile application to connect to the deadbolt, unlock it to open the door, and gets into the house.

[5]MSC (Message Sequence Chart) is a graphical language for the description of the interaction between different components of a system. This language is standardized by the ITU (International Telecommunication Union).

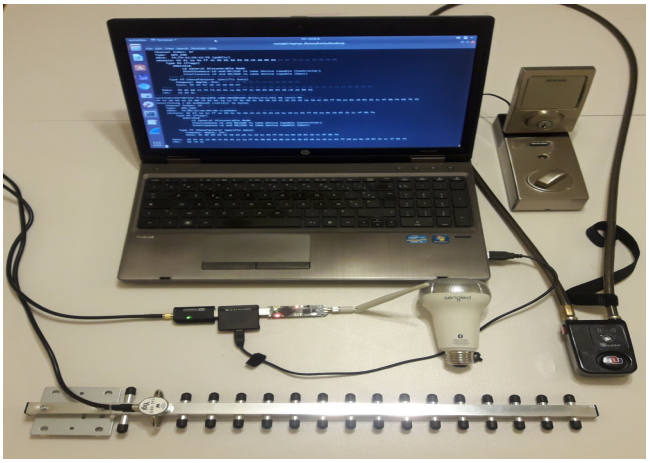

Fig. 4. Attacker environment: laptop (HP Probook 6560b), 25dBi directional Yagi antenna, Bluetooth dongle LM1010, Ubertooth One, Linux Ubuntu 16.04 LTS, Linux software utilities, smart deadbolt, bikelock, and a lightbulb.

Attacker environment. To generate attacks on BLE devices, we use different types of tools (viz., Fig. 4). We use three hardware components: an Ubertooth One [5], a Bluetooth dongle, and a high-gain antenna. Ubertooth One is a hardware component designed to perform Bluetooth experimentation, in particular, on Bluetooth low energy. The Bluetooth dongle is a dual band dongle (LM Technologies LM1010), which allows us to establish connections with Bluetooth smart devices. The antenna is a 2.4GHz Yagi antenna with a high gain of 25dBi. For software components, we use three software tools: *Ubertooth btle*, *hcitool*, and *gatttol*. The *Ubertooth btle* is a software utility that is used along with Ubertooth One to capture BLE traffic and follow an ongoing BLE communication. *Hcitool* is part of the Linux BlueZ Bluetooth package. It is used to communicate with the local Bluetooth controller of the computer and perform a multitude of operations, such as scanning for nearby Bluetooth devices. Finally, the *gatttol* software utility, which is used to establish connections with Bluetooth smart devices and send read and write commands to the devices to read and write the characteristics (values) of a given target service.

In the next subsection, we conduct attacks on the three Bluetooth smart devices, following the attack tree of Fig. 2. We perform attacks against data confidentiality, user authenticity, data integrity, and device availability. We present these attacks according to Stalling's attack classification [23]: Interception attacks cover attacks on data confidentiality, fabrication attacks contain attacks on user authenticity, modification attacks cover attacks on data integrity, and interruption attacks include

attacks on device availability. In this way, we cover all four fundamental security services, also known as information assurance pillars, defined in the DoD (U.S. Department of Defense) Information Assurance Certification and Accreditation Process [17] for a better security evaluation.

A. Bicycle lock

In this first use case, we consider the Bluetooth NULOCK bikelock from NuVending[6]. This lock is equipped with a braided steel cable of 47 inches, a 110dB alarm protection, and costs around $40. It runs Bluetooth v4.0+LE (Low Energy). It is controlled through a mobile application called Nulock (has 1,000+ downloads in Play Store), which allows the owner of the bike to lock and unlock the bikelock using BLE and by just standing close to the lock.

Interception attack. To intercept specific BLE traffic, we first start by detecting nearby Bluetooth smart devices, in particular, the bikelock. To this end, we use a Bluetooth dongle (LM Technologies LM1010) plugged in a laptop (HP Probook 6065b) that runs Linux (Ubuntu 16.04 LTS). Then, we use the *hcitool* software utility to communicate with the dongle and perform operations, such as scanning for nearby Bluetooth devices. We detect the bikelock and obtain both its address (`f8:36:9b:48:09:d9`) and user-friendly name (*smart-lock*). The *hcitool* command sends *scan_requests* to detect Bluetooth smart devices, which reply back by a *scan_response*. Next, we use Ubertooth One along with *Ubertooth-btle* software utility to eavesdrop any communication that involves the bikelock. Last, to display the captured packets and read their contents, we use *Wireshark*[7] with BLE packet dissector. We have activated the Bluetooth interface of the smartphone and have stood next to the lock. The connection between the smartphone and the bikelock has taken place and has been successfully intercepted by Ubertooth One, which has managed to intercept the *connect_request* packet (viz., Fig. 5), compute the hop sequence, and follow up the communication.

```
▾ Bluetooth Low Energy Link Layer
    Access Address: 0x8e89bed6
  ▸ Packet Header: 0x2245 (PDU Type: CONNECT_REQ, ChSel: #1, TxAdd: Random, RxAdd: Public)
    Initator Address: 6e:b2:de:c8:79:f9 (6e:b2:de:c8:79:f9)
    Advertising Address: TexasIns_48:09:d9 (f8:36:9b:48:09:d9)
  ▾ Link Layer Data
      Access Address: 0xaf9a83dd
      CRC Init: 0x7b7474
      Window Size: 3 (3.75 msec)
      Window Offset: 38 (47.5 msec)
      Interval: 39 (48.75 msec)
      Latency: 0
      Timeout: 2000 (20000 msec)
    ▸ Channel Map: ffffffff1f
      ...0 0101 = Hop: 5
      001. .... = Sleep Clock Accuracy: 151 ppm to 250 ppm (1)
  CRC: 0x368e58
```

Fig. 5. Wireshark view of the *connect_request* sent from the smartphone to the bikelock and intercepted by Ubertooth One.

We have executed some commands, such as changing the lock configuration using the mobile application. This has generated write-command packets sent from the smartphone to the bikelock. Ubertooth One has managed to follow up and intercept those packets. Interestingly, we have tried to interpret the content of one of the packets (viz., Fig. 6) and found a value (`a171727374313301`) being transmitted as a value for a characteristic which handle is `0x0025`. After decoding that value, it turns out that part of that value (underlined) corresponds to the ASCII code for the password `qrst13` which the user has initially set up. Now that the attacker knows the password, it can use it to bypass the authentication mechanism that is used in the mobile application and unlock the lock. The attacker just has to download the application on its smartphone, set up the password `qrst13`, stand near the bike, unlock the lock, and cycle the bike away. Also, the attacker can adopt a more sophisticated approach to eavesdrop the communication between the smartphone and the bikelock. For example, it can use a drone to fly over the bike while the owner is unlocking it. The drone intercepts the packets and flies back to the attacker. Notwithstanding, as drones make noises, another passive alternative consists of using a high gain directional antenna, such as the Bluegun [19], to increase the interception range. The attacker can stand far away from its victim while the latter is unlocking its bike and capture the packets without being noticed. We have actually performed this attack using a 5dBi omnidirectional antenna by standing **100 meters** away from the bikelock at Queen's university campus. Note that we have observed that this bikelock uses encryption only at the beginning when the owner sets up the initial password (probably to send the password to the bikelock). After that, all communications are in plaintext, including the password.

```
▾ Bluetooth Low Energy Link Layer
    Access Address: 0xaf9a83dd
    [Master Address: 6e:b2:de:c8:79:f9 (6e:b2:de:c8:79:f9)]
    [Slave Address: TexasIns_48:09:d9 (f8:36:9b:48:09:d9)]
  ▸ Data Header: 0x0f0e
    [L2CAP Index: 54]
  ▸ CRC: 0x8d731a
▸ Bluetooth L2CAP Protocol
▾ Bluetooth Attribute Protocol
  ▸ Opcode: Write Request (0x12)
  ▾ Handle: 0x0025 (Unknown: Unknown)
      [Service UUID: Unknown (0xffe0)]
      [UUID: Unknown (0xfff1)]
    Value: a171727374313301
```

Fig. 6. Wireshark view of a write-command packet intercepted by Ubertooth One during smartphone-bikelock communication.

Fabrication attack. Knowing the password and the Bluetooth device address, we have tried to connect and interact with the bikelock using the `gatttool` software utility that runs on a Linux system (Ubuntu 16.04 LTS) of a laptop. We have observed that the laptop gets paired with the bikelock using Bluetooth Just Works mode and gets disconnected after few seconds. This is mainly due to the application protocol that runs on the bikelock. The bikelock, as a BLE peripheral, expects the laptop, as a BLE central, to send some information to ensure that it is talking to the mobile application that was developed for communicating with the bikelock. This brings us to analyze the packets that have been captured by Ubertooth One and to find a pattern that would allow as to convince the bikelock that it is talking to the NULOCK mobile application. Interestingly, we have found that the mobile application sends a write request that contains a 20-byte value

[6]NuVending of NUNET is an American company specialized in designing and selling innovative IoT products.

[7]Wireshark is a network sniffer and protocol analyzer software utility.

(a13734313638390927802 6b1364b0770d6e237e). We have observed that the first most significant eight bytes (underlined) are fixed for each connection, whereas the remaining bytes vary by changing the password. Thus, we have tried to connect again to the bikelock and send a write command to the handle 0x0025 with the same twenty-byte value. We have observed that in this time the bikelock has not dropped out the connection. This means that the twenty-byte sequence is the right value to send in order to keep the connection with the bikelock running. Next, we have sent a write command to the handle 0x0025 with the learned password and Poof! the bikelock got unlocked.

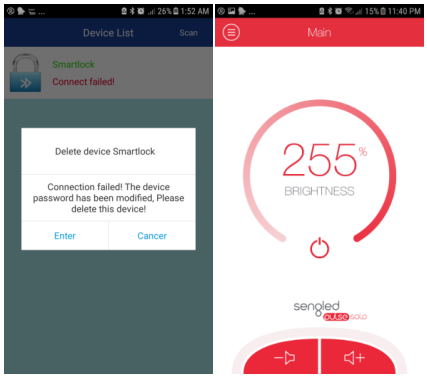

Fig. 7. Mobile application interfaces after BLE attacks (from left to right): (a) Nulock for the bikelock and (b) Pulse for the lightbulb.

Modification attack. We have thought of the possibility of illegally changing the configurations of the bikelock. One important configuration is the password. We have set up Ubertooth One to follow the connection between the bikelock and the legitimate user while changing the password using the mobile application. After we have changed the password, we have analyzed the packets and found a write command that contains both passwords, the old one and the new one, separated by one byte (0x07). Thus, we have paired the laptop again with the bikelock, sent the twenty-byte sequence to hold the connection, and sent a slightly modified write command to change the password. We have just swapped the old password with the new one in such a way so that the new password becomes qrst13. The bikelock has replied with a message indicating that the command was successfully executed. This means that the integrity of the stored data in the device has been successfully corrupted.

Interruption attack. In order to check the impact of the write command that we have sent during the modification attack, we have used the mobile application and tried to legally connect the bikelock and unlock it. The mobile application has popped up a message informing that the password has been changed and that the bikelock has to be removed from the list as illustrated in Fig. 7.(a). In this way, the legitimate user has been locked out from its bike. If the user forces the lock, the lock will trigger its 110dB alarm. The alarm will not stop until it runs out of battery, which will take hours. At this point, the bike lock requests the user to input the new password which is only known by the attacker. This approach can be used to set up a ransomware that performs the same attack and locks users out and forces them to pay a certain amount of money to unlock their items.

Furthermore, to evaluate the impact of connection deprivation discussed in Section V, we have established a connection with the bikelock and used that 20-byte value to hold the connection. Next, using the legitimate application, we have tried to connect to the bikelock which failed as shown in Fig. 8.(a). An attacker standing far away from the bike, can perform this attack using a high-gain directional antenna and deprive a legitimate user from unlocking its bike.

B. Smart lightbulb

In this second case, we consider the Bluetooth pulse solo lightbulb from Sengled[8]. The lightbulb comes with an integrated JBL speaker to allow users to broadcast audio data on the lightbulb from their smartphones. It costs around $70. It runs Bluetooth 4.0+LE (Low Energy). Users can download and install a mobile application called Pulse (has 100,000+ downloads in Play Store) to connect their smartphones to the lightbulb and control it remotely. They can use the application to switch the lightbulb ON and OFF, and adjust its brightness.

Interception attack. Similar to the bikelock, we have started by detecting the lightbulb and learning its Bluetooth device address (08:7c:be:36:e0:49) as well as its user-friendly name (C01-A66_Pulse Solo) using hcitool utility. The lightbulb periodically broadcasts advertisement packets on channel 37 to indicate its presence in the neighborhood. The mobile application, i.e., Pulse, does not require any password from the user. Thus, this time we do not have to intercept any confidential information, such as a password. Still, we are interested in intercepting write-commands that the application sends to change the status and configurations of the bulb such as the commands that modify the brightness of the lightbulb and the ones that turn the light ON and OFF. We have generated commands to switch the light ON and OFF and to modify the lightbulb brightness. At the same time, we have set up Ubertooth One along with *Ubertooth-btle* software to intercept the traffic that is generated by the commands that are executed from the smartphone. We have captured the packets that were used to modify the brightness of the lightbulb. Those packets carry write-commands to the handle 0x0017 with a 168-bit value (7efeffffff000000000100010000000ff64400007e). The byte that contains the value 64 (underlined) represent the brightness, which is equal to 100% in this case. This is the same write-command that is used to switch the light ON. Therefore, when switching the lightbulb OFF, there will be a write-command to the handle 0x0017 with a value equal to 7efeffffff000000000100010000000ff00400007e.

Fabrication attack. The Pulse application does not implement any authentication mechanism. Anybody can install the application and connect to the lightbulb and remotely switch it ON

[8]Sengled is an international company specialized in smart lighting products.

and OFF. An attacker can take over the lightbulb by connecting to it and illegally executing commands, such as switching the light ON and OFF and changing the light brightness.

Modification attack. To breach the integrity of the lightbulb, we have tried to illegally change the configurations of the lightbulb. Knowing the value to be written on the handle `0x0017`, we have connected the laptop to the lightbulb and have executed write-commands on the bulb. By setting the brightness byte to `0x00`, we have managed to turn the lightbulb OFF, and by setting it to `0x64` we have turned the lightbulb ON with 100% brightness. Interestingly, since the highest value to be represented on one byte is 255, we have tried to execute a write-command with a value set to `0xff`. The command was executed with success. Then, to check the impact of the last command, we have used the mobile application to connect to the lightbulb and found out that the current displayed brightness is 255% as illustrated in Fig. 7.(b). Hopefully, nothing dangerous has happened after we have boosted the brightness beyond its maximum for few seconds. Only God knows, what could have happened to the bulb if we have left it with that brightness for a longer time.

Interruption attack. In this scenario, there are three ways to cause a denial of service on the lightbulb: (1) Sending a steady-stream of write-commands to the handle `0x0017` with a value set to `0x00` to switch OFF the light definitely. (2) Sending a steady-stream of write-commands to the handle `0x0017` with a value set to `0xff` and see what will happen after sometime. (3) Establish an illegal connection with the lightbulb and deprive legitimate users from connecting to it and using it. We have performed this third attack option and tried to connect using the application as legitimate users, but the application popped up the message shown in Fig. 8.(b). We have installed the lightbulb inside an office in the 6^{th} floor of a building at Queen's university campus. By staying **85 meters** away outside the building inside a car while it was raining, we have managed to take over the lightbulb. We note that in the lightbulb's manual, it is mentioned that the lightbulb has an operational range of **10 meters**.

C. Smart deadbolt

In this last case, we consider the Bluetooth Sense deadbolt from Schlage[9]. The deadbolt is embedded with Wi-Fi as well as Bluetooth smart technology. It costs around $250. It runs Bluetooth 4.1+LE (Low Energy). Users can download and install a mobile application called SchlageHome (has 50,000+ downloads in Play Store) to connect their smartphones to the deadbolt and unlock it. The deadbolt can also be connected to the Internet through its Wi-Fi interface and get unlocked from anywhere through Internet.

Interception attack. We have captured the communication between this home lock and the smartphone using Ubertooth One. Then, using *Wireshark* we have displayed the packets that have been intercepted and tried to interpret them. It turns out that this deadbolt uses AES-encryption with data freshness. This means that the packets can neither be read (decrypted) nor replayed. We could not apply simple attacking tools to disclose any confidential information. We have also tried to capture the pairing messages in an attempt to crack the LTK key, but have observed that the pairing is actually not happening over BLE. We have noticed that the application requires the Wi-Fi interface to remain active while the pairing is performed. We have concluded that this deadbolt uses the Out-of-Band pairing mode to accomplish a secure pairing. We have also reverse-engineered the mobile application to understand its code, but the code was too complex and appeared to be well designed and implemented. Hence, this deadbolt requires more sophisticated approaches that we leave for future work.

Fabrication attack. We have tried to replay the packets that we have captured and observed that the deadbolt cut off the connection right after receiving our replayed packets. This means that the deadbolt is implementing a kind of replay protection to discard any replayed or unexpected packets.

Modification attack. We have observed that the deadbolt discards most of the packets that we have replayed and injected. However, we have managed to successfully change the values of some characteristics, such as the one identified by the handle `0x0023`. The impact of changing the value of the characteristic `0x0023` is discussed in the next paragraph.

Interruption attack. Although this deadbolt has proven to have security implemented, it does not resist against the connection deprivation attack discussed in Section V. As the deadbolt accepts only one connection at a time, an attacker can easily occupy the connection and take over the deadbolt. Hence, any legitimate user who tries to connect and unlock the deadbolt to get home, will be locked out. We have generated this attack and tried to connect using the legitimate application. The latter has displayed the screenshot shown in Figure 8.(c). Nevertheless, we have observed that the connection is dropped after 60 seconds. We have tried to understand the reason behind that and have found that the deadbolt periodically disconnects from the application after each 60 seconds and then reconnects. We have considered that as a security measure against eavesdroppers since re-establishing a new connection results in a new hop sequence. Thus, to perform the connection deprivation attack, the attacker needs to reconnect after each 60 seconds. However, by applying some fuzzing, we have found that we can successfully change the value of the handle `0x0023`, and that if we change that value to `0xff` few seconds before those 60 seconds elapse, we can keep the connection for another minute. Hence, the attacker just has to send a write command to the handle `0x0023` with a value `0xff`, say each 55 seconds, to not get disconnected and indefinitely deprive legitimate users from unlocking the deadbolt. We have used our high-gain antenna and managed to successfully perform this attack from inside a car that we have parked **425 meters** away from the deadbolt. We emphasize

[9]Schlage is an American lock manufacturer founded in 1920. It produces high-security key and cylinder lines, Primus, Everest, and Everest Primus XP.

that it was raining when we performed this attack from that distance.

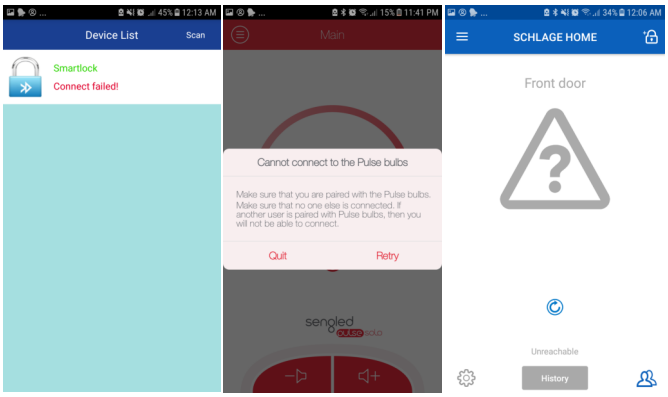

Fig. 8. Mobile application interfaces after BLE connection deprivation (from left to right): (a) Nulock for the bikelock, (b) Pulse for the lightbulb, and (c) SchlageHome for the deadbolt.

Countermeasure. As a mitigation to the connection deprivation attack, Bluetooth smart devices should be designed to accept multiple connections at a time. This will for example allow two or more users to control a device at the same time. Notwithstanding, we boldly forewarn that connection handling must be implemented correctly to avoid falling back into the connection dumping vulnerability (Bluecutting attack) introduced by Lounis *et al.* in [15], where an attacker A spoofs a legitimate user U to connect to device D, while the legitimate user U is already connected to D, causing the disconnection (hijacking) of D from the legitimate user U.

VII. Conclusion

BLE (Bluetooth Low Energy) is being embedded in many smart devices of different application fields, such as medical, home automation, transportation, and security devices. However, BLE security has been left for application developers and device vendors. As most vendors do not want to pay for security, there is a large number of vulnerable and expensive smart devices strolling around in the market. In this paper, we have discussed BLE security, in particular, when the Just Works is used. We have shown through a practical case study of three different Bluetooth smart devices, how vulnerable certain devices are. We have also discussed the connection deprivation vulnerability that can be exploited to deny legitimate users from using their own devices and proposed a mitigation technique to this attack as well.

We cannot force Bluetooth smart device vendors to invest for security. Still, we can warn users about the possible risk of buying insecure devices and choosing convenience over security, privacy, and safety. Bluetooth smart devices are predicted to be nearly one-third of the forty-eight billion IoT (Internet of Things) devices in 2021. If BLE security is not considered seriously, by 2021 we will have billions of vulnerable devices in the market. Those devices will be used in IoT applications and make IoT vulnerable as well.

Acknowledgment

This work is partially supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Canada Research Chairs (CRC) program.

References

[1] Bluetooth-SIG. "Bluetooth Core Specification Version 5.0," Bluetooth Spec document, 2018.
[2] W. Diffie and M. E. Hellman. "New Directions in Cryptography," IEEE Transaction Information Theory, vol. 22, no. 6, pp. 644-654, 1976.
[3] NIST. "Advanced Encryption Standard (AES)," http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf, 2001.
[4] D. Spill. "Bluetooth Packet Sniffing Using Project Ubertooth," http://2012.ruxcon.org.au/assets/rux/Spill-Ubertooth.pdf, 2012. (Accessed: 06-14-2019).
[5] Great-Scott-Gadgets. "Ubertooth One: A Wireless Development Platform Suitable for Bluetooth Experimentation," https://greatscottgadgets.com/ubertoothone/, 2009. (Accessed: 06-14-2019).
[6] M. Ryan. "I am Jack's Heart Monitor," http://lacklustre.net/bluetooth/hacking_btle-i_am_jacks_heart_monitor-mikeryan-toorcon_2012.pdf, 2012 (Accessed: 06-14-2019).
[7] A. Ray, and V. Raj, and M. Oriol, and A. Monot, and S. Obermeier. "Bluetooth Packet Sniffing Using Project Ubertooth," In the IEEE 11th International Conference on Software Testing, Verification and Validation, pp. 384-393, 2018.
[8] A. Rose, and B. Ramsey. "Picking Bluetooth Low Energy Locks from a Quarter Mile Away," https://www.defcon.org/html/defcon-24/dc-24-speakers.html, 2016. (Accessed: 06-14-2019).
[9] Q. Zhang, and Z. Liang. "Security Analysis of Bluetooth Low Energy Based Smart Wristbands," In the 2nd International Conference on Frontiers of Sensors Technologies, 2017.
[10] B. Cyr, and W. Horn, and D. Miao, and M. Specter. "Security Analysis of Wearable Fitness Devices (Fitbit)," MIT report, 2014.
[11] O. Arias, and J. Wurm, and K. Hoang, and Y. Jin. "Privacy and Security in Internet of Things and Wearable Devices," In the IEEE Transactions on Multi-Scale Computing Systems, vol. 1, No. 2, pp. 99-109, 2015.
[12] H. C. Chen, and M. A. A. Faruque, and P. H. Chou. "Security and Privacy Challenges in IoT-based Machine-to-Machine Collaborative Scenarios," In the International Conference on Hardware/Software Codesign and System Synthesis, pp. 1-2, 2016.
[13] A. Hilts, and C. Parsonsm, and J. Knockel. "Every Step You Fake: a Comparative Analysis of Fitness Tracker Privacy and Security," https://openeffect.ca/fitness-trackers/, 2016. (Accessed: 06-14-2019).
[14] P. Gullberg. "Denial of Service Attack on Bluetooth Low Energy," https://mafiadoc.com/queue/denial-of-service-attack-on-bluetooth-low-energy_59e9aa541723dd2ddf0d7842.html, 2016.
[15] K. Lounis, and M. Zulkernine. "Connection Dumping Vulnerability Affecting Bluetooth Availability," In the Proceedings of the 13th International Conference on Risks and Security of Internet and Systems, no. 11391, pp. 188-204, 2018.
[16] B. Schneier. "Attack Trees: Modeling Security Threads," https://www.schneier.com/academic/archives/1999/12/attack_trees.html, 1999. (Accessed: 06-14-2019).
[17] DoD, "Information Assurance: Instruction-8500.1," Assistant secretary of Defense for Network and Information integration, Jones & Bartlett Learning, 2002.
[18] A. M. Lonzetta, and P. Cope, and J. Campbell, B. J. Mohd, and T. Hayajneh, "Security Vulnerabilities in Bluetooth Technology as Used in IoT," In Journal of Sensor and Actuator Networks, doi:10.3390/jsan7030028, 2008.
[19] J. Hering. "Bluetooth Cracking Gun: BlueSniper," https://www.defcon.org/html/links/dc-archives/dc-12-archive.html, 2004. (Accessed: 06-14-2019).
[20] S. Jasek. "Blue Picking: Hacking Bluetooth Smart Locks," HITBSecConf, 2017.
[21] V. Tan. "Hacking BLE Bicycle Locks For Fun and a Small Profit," DEF CON 26, 2018.
[22] D. Cauquil. "BtleJuice: The Bluetooth Smart Man In The Middle Framework," Hack.lu, 2016.
[23] W. Stalling. "Cryptography and Network Security: Principles and Practice : Instructor's Manual," Prentice Hall, 1998.