

InjectaBLE: Injecting malicious traffic into established Bluetooth Low Energy connections

Romain Cayre, Florent Galtier, Guillaume Auriol, Vincent Nicomette, Mohamed Kaâniche, Géraldine Marconato

▶ **To cite this version:**

Romain Cayre, Florent Galtier, Guillaume Auriol, Vincent Nicomette, Mohamed Kaâniche, et al.. InjectaBLE: Injecting malicious traffic into established Bluetooth Low Energy connections. IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2021), Jun 2021, Taipei (virtual), Taiwan. 10.1109/DSN48987.2021.00050 . hal-03193297v2

HAL Id: hal-03193297

https://laas.hal.science/hal-03193297v2

Submitted on 12 Apr 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

InjectaBLE: Injecting malicious traffic into established Bluetooth Low Energy connections

Romain Cayre*‡, Florent Galtier*, Guillaume Auriol*†, Vincent Nicomette*†, Mohamed Kaâniche*, Géraldine Marconato‡
*CNRS, LAAS, 7 avenue du colonel Roche, F-31400
†Univ de Toulouse, INSA, LAAS, F-31400
‡APSYS.Lab, APSYS
Email: *firstname.lastname@laas.fr ‡firstname.lastname@airbus.com

***Abstract*—Bluetooth Low Energy (BLE) is nowadays one of the most popular wireless communication protocols for Internet of Things (IoT) devices. As a result, several attacks have targeted this protocol or its implementations in recent years, illustrating the growing interest for this technology. However, some major challenges remain from an offensive perspective, such as injecting arbitrary frames, hijacking the *Slave* role or performing a Man-in-The-Middle in an already established connection. In this paper, we describe a novel attack called *InjectaBLE*, allowing to inject malicious traffic into an existing connection. This attack is highly critical as the vulnerability exploited is inherent to the BLE specification itself, which means that any BLE connection can be possibly vulnerable, regardless of the BLE devices involved in the connection. We describe the theoretical foundations of the attack, how to implement it in practice, and we explore four critical attack scenarios allowing to maliciously trigger a specific feature of the target device, hijack the *Slave* and *Master* role or to perform a Man-in-the-Middle attack. Finally, we discuss the impact of this attack and outline some mitigation measures.**

***Index Terms*—injection, IoT, Bluetooth Low Energy, hijacking, Man-in-the-Middle**

I. Introduction

Nowadays, Internet of Things (IoT) devices are everywhere: many objects of our daily lives, from fridges to watches, now incorporate micro-controllers and modems, allowing them to communicate with their environment and to offer new services. Several wireless communication protocols have been developed in recent years to implement these services, among them the Bluetooth Low Energy (BLE) protocol. BLE provides a lightweight protocol stack and allows devices to communicate easily and reliably with a minimal energy consumption, which fits perfectly the constraints of connected objects. It is also widely deployed in smartphones, computers and tablets, enabling direct communications without the need for additional gateways in the network. As a result, many IoT devices already rely on BLE to communicate with their environment.

The growing interest for this technology also raises legitimate concerns about the security of BLE. In the recent years, the security of this protocol has been actively studied both from an offensive and a defensive perspective, highlighting serious flaws in its specification [5] and in various implementations. Some papers focused on eavesdropping a BLE connection, which is not straightforward because of the use of a channel hopping algorithm, while other papers described active attacks such as jamming, hijacking or Man-in-the-Middle attacks. However, to our knowledge, all state of the art offensive techniques described so far require the attack to be carried out before the targeted BLE connection is established, or are based on highly invasive techniques such as jamming. Even if some papers mentioned a theoretical non invasive injection-based attack in an established connection [19] or consider it difficult to achieve [21], it has never been implemented in practice and the new offensive capabilities provided by this strategy have not been studied.

In this paper, we demonstrate the practical feasibility of such attacks, which increases significantly the attack surface of the BLE protocol. We present a novel approach named *InjectaBLE* allowing to perform an arbitrary frame injection into an already established BLE connection. We first explain its theoretical foundations, and then present various experiments illustrating its feasibility.

Four critical offensive scenarios that take advantage of this injection attack are investigated: we show that an attacker could use our approach to stealthily trigger a specific feature of a device, hijack any role involved in the targeted connection or perform a Man-in-the-Middle attack during the connection. We demonstrate that most of these scenarios, that were considered unrealistic until now, are in fact quite easy to perform and could have serious consequences on the security of any BLE device compliant to Bluetooth Core Specification irrespective of how it is implemented. We finally discuss the impact of this attack and potential mitigation measures.

In summary, the major contributions of the paper are:

- the presentation of a novel injection-based attack into an already established BLE connection, from its theoretical foundations to its practical implementation, the objective being to increase awareness of the vulnerability of this widely used wireless protocol and the potential threats if appropriate security measures are not applied.
- a sensitivity analysis, allowing to understand the impact of three key parameters on the injection success.
- four critical attack scenarios based on the injection attack, allowing to maliciously trigger a specific feature of a device, hijack the role of any device involved in the connection and perform a Man-in-the-Middle attack during an established connection.

- the proposal of counter-measures to mitigate this attack.

The paper is organised as follows. Section II presents the state of the art of offensive security targeting the BLE protocol and clearly outlines the innovative contribution of this work. Section III presents an overview of the BLE protocol with a specific focus on the Link Layer, introducing some key concepts that are necessary to understand the attack strategy. Section IV presents an overview of the attack and introduces the main challenges that must be addressed for this attack to be successful. Then, Section V describes the theoretical foundations of our attack and its practical implementation. Section VI shows how this attack can be used in four critical attack scenarios, while section VII presents a set of experiments carried out to analyse the impact of three main parameters regarding the attack success. Section VIII proposes multiple counter-measures that could be used to mitigate the impact of our attack or detect it and Section IX concludes the paper.

II. Related work

In the past few years, multiple attack strategies and tools targeting the BLE protocol have been released.

Sniffing a Bluetooth Low Energy connection is a non-trivial task, because of the channel hopping algorithm used by the devices when they are in *connected mode*. In [19], M. Ryan demonstrated that a specific connection can be easily sniffed if the sniffer successfully receives the packet initiating the connection which includes the initial channel hopping parameters. He also showed that an attacker may be able to retrieve the parameters of an already established connection by monitoring specific events. This approach was then improved by D. Cauquil in [8] to infer the channels to listen to. In [10], he also adapted the sniffing strategy to deal with a new algorithm based on a pseudo-random generator that has been introduced in the BLE 5.0 specification [5], called *channel selection algorithm #2*. Finally, a new tool named *Sniffle* has also been released [17] by S. Qasim Khan. It provides interesting features such as support for the new physical layers introduced in the BLE 5.0 specification or a mode allowing to follow the target device hopping along the advertising channels. Since these channels are used to broadcast data and indicate the presence of a specific device, the probability of a successful sniffing is increased.

Multiple active attacks have also been presented in recent years. First, jamming-based attacks have been explored by Brauer et al. in [6], they demonstrated an attack allowing to selectively jam advertisements. D. Cauquil also presented a new offensive tool named *BTLEJack* [9] allowing to disrupt an existing connection by jamming packets transmitted by one of the devices involved in a connection, called *Slave*. The direct consequence of this jamming strategy is a disconnection of the other device, named *Master*, allowing the attacker to synchronise with the *Slave* instead of the legitimate device, resulting in hijacking the *Master* role during an established connection. However, this strategy cannot be used to hijack the *Slave* role, which could also be relevant from an offensive perspective, and, being based on a jamming technique, is highly invasive and visible.

Second, two major tools, *GATTacker* [15] by S. Jasek and *BTLEJuice* [7] by D. Cauquil, can be used to perform a *Man-in-the-Middle* attack. *GATTacker* clones the advertisements transmitted by the target device (called *Peripheral*) to indicate its presence and tries to advertise them faster, forcing the device initiating the connection (also known as *Central*) to connect on a cloned *Peripheral* controlled by the attacker. The approach adopted by *BTLEJuice* directly establishes a connection with the target *Peripheral*, forcing it to stop advertising, then it exposes a cloned *Peripheral* to the *Central*. Both of these strategies are based on advertisements spoofing: as a consequence, they can only perform a Man-in-the-Middle attack if the connection is not already established.

Multiple studies have also addressed the security of authentication and encryption mechanisms in BLE connections. In 2013, M. Ryan presented *CRACKLE* [20], a tool exploiting a weakness in the first version of the BLE pairing process to quickly bruteforce the keys involved in the BLE *connected mode* security. In [1], Antonioli et al. introduced an attack named *KNOB* (Key Negotiation of Bluetooth), to downgrade the key entropy from 16 to 7 bytes, which drastically reduces the attacker's effort to bruteforce the key. In [2], they also analysed the Cross-Transport Key Derivation, a mechanism allowing to share keys between Bluetooth Classic and BLE, and demonstrated four attacks named *BLUR attacks* abusing this feature, allowing to impersonate a device, manipulate traffic or establish a malicious session. Similarly, Wu et al. demonstrated *BLESA* [23], an active attack abusing the reconnection process of an already paired *Central* to impersonate the corresponding *Peripheral* and transmit some unencrypted spoofed data. Von Tschirschnitz et al. presented a method confusion attack [22] aiming at forcing the pairing of two devices using different methods. While some of these attacks can be used to impersonate a device, none of them can hijack such a device during an established BLE connection.

Previous research have also focused on discovering vulnerabilities that are linked to the stack implementation rather than the protocol specification, such as *Blueborne* [3] in 2017 or *BleedingBit* [4] in 2018. Also, in [14], Garbelini et al. presented a fuzzing framework named *SweynTooth* targeting various BLE stacks, discovering a dozen of vulnerabilities. While their consequences are generally severe, they are related to specific implementations and cannot be generalised.

Limitations of existing approaches and contribution. As far as we know, none of the existing research in this field have focused on injecting malicious frames into an existing connection. In [21], Santos et al. hypothesised it would be too difficult to set up such an injection-based attack in BLE, because they considered that dealing with race conditions is complex and would require high performance hardware. As a consequence, they rejected the possibility of such an approach. However, as we will further demonstrate afterwards and illustrate it experimentally, such an attack is actually possible, and can even take advantage of the race condition

Santos et al. stated as a limitation to injection-based attacks. We also demonstrate that this approach can be used to perform new attack scenarios that haven't been explored yet, such as hijacking the *Slave* role or performing a Man-in-the-Middle attack during an already established connection.

III. Bluetooth Low Energy

This section presents a brief overview of the BLE protocol as well as some more detailed descriptions of the Link layer (LL), which are directly related to our injection attack.

A. Overview

Bluetooth Low Energy is a lightweight variant of Bluetooth, dedicated to devices needing low energy consumption.

The stack is split into two major parts: the *Controller* and the *Host*. The lowest layers are included in the *Controller*, while the highest ones are handled by the *Host*.

The physical layer is based on a *Gaussian Frequency Shift Keying* modulation. Three main modes can be used in BLE: an uncoded physical layer with a bitrate of 1 Mbit/s or 2 Mbits/s (respectively called *LE 1M* and *LE 2M*), or a coded physical layer using a 250 kbits/s or 500 kbits/s bitrate (called *LE Coded*). BLE operates in the ISM band from 2.4 to 2.5 GHz, and defines 40 channels, each with a bandwidth of 2 MHz. Three channels (37, 38 and 39) are dedicated to the *advertising mode* (allowing devices to broadcast data using some packets named *advertisements*), while the 37 others channels (numbered from 0 to 36) are dedicated to the *connected mode*, which is used when a connection is established between two devices.

Every BLE-based application using the *connected mode* is built on top of the *ATT* and *GATT* layers. These layers define a client / server model, providing a generic solution to exchange data between devices. Specifically, an *ATT* server is a database of *attributes*. Each *attribute* is composed of an identifier, a type and a value. An *ATT* client is able to interact with this database using some requests: for example, a *Read Request* allows the client to read a given *attribute*, while a *Write Request* allows to modify the value of an *attribute*. The *GATT* level provides an additional layer of abstraction to define some *services* including *characteristics* and creates generic profiles for a given type of device.

The *Security Manager* provides a set of *pairing* and *bonding* procedures to negotiate multiple keys dedicated to increase the security level of the connection. One of the most important keys is the *Long Term Key*, which allows to establish an AES-CCM encryption over the Link Layer to avoid eavesdropping. The *Generic Access Profile* (*GAP*) introduces four different roles, describing the device's behaviour. Regarding the *connected mode*, two roles are defined. The *Peripheral* role corresponds to a device that can transmit advertisements and is connectable, while the *Central* role corresponds to a device that can receive advertisements and establish a connection with another device. The *Peripheral* is also called *Slave* as it plays a slave role in a BLE connection; the *Central* is called *Master*.

B. Link layer internals

Our injection-based attack mainly relies on the exploitation of some specific features of the Link Layer. This subsection provides a detailed description of these features.

1) Frame format: Every BLE frame transmitted using the *LE 1M* mode is based on the format described in table I:

TABLE I: Frame format for *LE 1M*

Preamble	**Access Address**	**Protocol Data Unit (PDU)**	**CRC**
1 byte	4 bytes	variable	3 bytes

The preamble is used by the receiver to detect the start of a BLE frame. The *Access Address* indicates the mode in use, either *advertising mode* or *connected mode*. The *Protocol Data Unit* is a variable field containing the data to transmit. Finally, a 3 bytes CRC is used for integrity checking.

2) Initiating a connection: When a *Peripheral* is not in a connected state, it broadcasts some advertisements on the advertising channels. The payload generally includes some information allowing to identify the device, such as the device name. To establish a connection with a *Peripheral*, the *Central* transmits a dedicated type of advertisement named *CONNECT_REQ* right after the reception of an advertisement from the *Peripheral*. The corresponding *LL PDU*, described in table II, includes some parameters used during the connection. The *Access Address* field is used by both devices for every frame transmitted during the connection.

3) Channel selection: The *Channel Map* and *Hop Increment* fields (cf. Table II) are used by the channel selection algorithm. Indeed, a BLE connection uses a channel hopping mechanism to avoid interference with other BLE connections or wireless communication protocols. Two main channel selection algorithms are currently usable: *Channel Selection Algorithm #1* is based on a simple modular addition, while *Channel Selection Algorithm #2* is based on a pseudo-random generator. Both of them can be predicted by an attacker to sniff an established connection (see [19] and [10]). In our study we consider *Channel Selection Algorithm #1*, which is the most commonly used algorithm, however the proposed approach can be easily adapted to the second algorithm.

4) Transmit window: Two fields *WinSize* and *WinOffset* (cf. Table II) are used to define the *transmit window*. Indeed, the first frame of the connection is transmitted on the first selected channel by the *Central* to the *Peripheral* at time t_0 during the *transmit window* defined by formula 1:

$$\begin{cases} t_{start} \leq t_0 \leq t_{start} + d_{size} \\ t_{start} = t_{init} + 1250\mu s + d_{offset} \end{cases} \quad (1)$$

With t_{init} the end of transmission time of the *CONNECT_REQ* frame, $d_{offset} =$ *WinOffset* $\times 1250\mu s$ and $d_{size} =$ *WinSize* $\times\ 1250\mu s$.

t_0 indicates the beginning of the first *connection event*, and is used as a time reference for next *connection events*.

TABLE II: CONNECT_REQ PDU

Init. addr.	Adv. addr.	Access addr.	CRCInit	WinSize	WinOffset	Hop interval	Latency	Timeout	Channel Map	Hop Increment	SCA
6 bytes	6 bytes	4 bytes	3 bytes	1 byte	2 bytes	2 bytes	2 bytes	2 bytes	5 bytes	5 bits	3 bits

5) Connection events: Let us consider a *connection event* that starts at the time t_n of frame transmission from the *Master* to the *Slave*, called the *anchor point*. t_0 corresponds to the first *anchor point*. When the *Slave* receives the frame, it waits during the *inter-frame spacing* ($150\mu s$) before sending a frame to the *Master*. A bit named *More Data* (*MD*) in the header of frames allows to indicate that more data is available and will be transmitted during the *connection event*. If the device does not have data to transmit, it will transmit an empty frame.

The time between two consecutive *anchor points* is given by the *Hop Interval* parameter, according to the formula 2:

$$d_{connInterval} = \textit{HopInterval} \times 1250\mu s \tag{2}$$

Each time a *connection event* is closed, the next channel is selected according to the channel selection algorithm in use. Each *connection event* is also identified by a 16-bit unsigned integer named *connection event count*. Figure 1 illustrates two typical consecutive *connection events*.

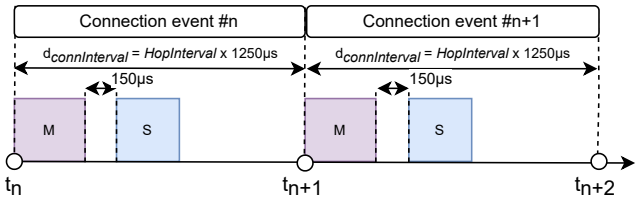


Fig. 1: Two consecutive *connection events*

6) Acknowledgement and flow control: Each BLE frame transmitted during a connection includes two 1 bit fields in the header of the *LL PDU*, indicating respectively the *Sequence Number* (*SN*) and the *Next Expected Sequence Number* (*NESN*). Each device also has two 1 bit counters, respectively named *transmitSeqNum* and *nextExpectedSeqNum*. The *transmitSeqNum* counter is incremented by one (modulo 2) if the previously transmitted data have been acknowledged. The *nextExpectedSeqNum* is incremented by one (modulo 2) when the next expected frame has been received.

7) Updating the parameters during the connection: The BLE protocol provides possibilities to update the parameters used by the channel selection algorithm. A *Master* is generally able to manage multiple connections simultaneously, and may need to modify a connection in order to optimise the following of multiple connections. It may also consider a given channel noisy due to high frame loss rate during transmission on that channel and may choose to blacklist it (i.e. mark it as unused). The Link Layer provides two main control frames, *CONNECT_UPDATE_IND* and *CHANNEL_MAP_IND*, to update the *Hop Interval* and the *Channel Map* respectively.

These frames include the new value of the field to update, and a two bytes field named *instant*. When the *Slave* receives one, it starts the corresponding procedure, and waits for the time when *instant* equals to *connection event count*. Then:

- In the case of a *connection update*, a *transmit window* similar to the one in the initiation of the connection is computed from the *WinOffset* and *WinSize* values of the *CONNECT_UPDATE_IND* frame. The new interval is then applied to the next *connection events*, as shown in Figure 2.
- In the case of a *channel map update*, the new *channel map* is used for next *connection events*.

8) Slave latency: The slave *latency* field (cf. Table II), that is initially proposed by the *Master* in the *CONNECT_REQ* packet and can be updated in a *connection update procedure*, allows the *Slave* to avoid entering the listening mode at every *connection event* in order to decrease its energy consumption.

IV. Adversary model and attack overview

This section presents a novel type of attack targeting BLE protocol, allowing the injection of arbitrary frames into an established connection. As seen in Section III, the BLE protocol provides a *connected mode*, allowing the involved devices to communicate only at some specific time, making injection-based attacks difficult to perform by design. According to the specification [5], one of the involved devices can expand the receiving window to compensate clock inaccuracy. However, this also opens the possibility for an attacker to abuse this feature by performing a race condition attack (see Figure 3). We focused our work on analysing the feasibility of such an injection, and explored techniques allowing to solve the following technical challenges:

- **(C1) identify when a malicious frame could be injected,**
- **(C2) investigate how to inject a malicious frame without altering the connection state consistency,**
- **(C3) check if the attack is successful or not.**

From an offensive perspective, the attack presented in this paper has a significant impact: indeed, although several attacks targeting BLE security have already been investigated in several studies, none of them have made it possible to interfere with an established connection without breaking the communication, at least for one of the concerned devices. The results presented in this paper show that such an attack is possible and can then be used to perform a wide set of critical offensive scenarios, including an illegitimate use of victim device features and hijacking attacks. We believe that this new offensive capability may consequently impact the availability, confidentiality and integrity of any BLE communication. Indeed, the vulnerability presented in this paper is related to the receiving window expansion described in the protocol specification, so any BLE device is potentially vulnerable, independently of its stack implementation. The threat is all

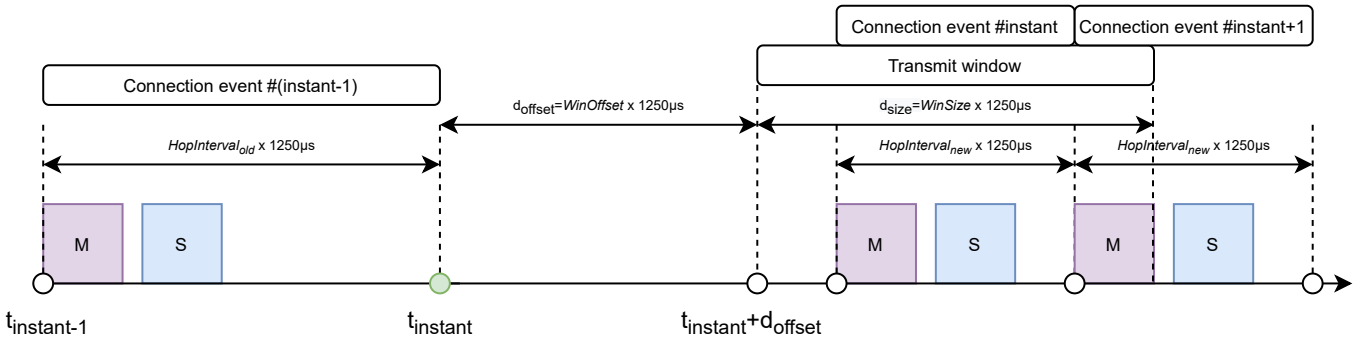


Fig. 2: Connection update procedure

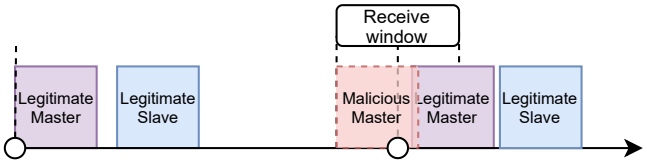


Fig. 3: Attack overview

the more serious as the attack is straightforward on common BLE chips and can be performed as soon as an attacker is within radio range of the targeted connection. The attack is also compatible with all versions of BLE, from 4.0 to 5.2. The adversary model considered is as follows:

- the attacker must be within the radio range of the target,
- the attacker uses a standard BLE 4.0 or BLE 5.0 device,
- the attacker is capable of passively sniffing the traffic, and actively crafting and transmitting spoofed packets on BLE channels,
- the attacker does not need to exploit any BLE vulnerability on the target devices.

As far as encrypted communications are concerned, the vulnerability being related to the design of the BLE Link Layer, it is independent of the security mechanisms provided by the protocol. Therefore, exploiting the race condition to inject a frame in an encrypted connection remains technically possible. Indeed, even if the attacker cannot obtain the Long Term Key used for encryption by some other mean, he can still inject an invalid packet, leading to a denial of service. As a consequence, enabling the security mechanisms provided by BLE limits the impact of the attack but the vulnerability itself (race condition allowing to inject a frame) remains, with at least an impact on availability.

V. InjectaBLE: injecting arbitrary frames in an established connection

In this section, we present the *InjectaBLE* attack, allowing to inject arbitrary frames in an established connection. Performing such an attack requires to identify a specific time when a frame can be successfully injected by the attacker, called the *injection point*. Subsections V-A and V-B describe the specific features of the Link Layer that make it possible to find such an injection point (challenge **C1** of Section IV). Subsection V-C describes how to inject the well-formed frame without altering the consistency of the connection state (challenge **C2**) and Subsection V-D describes how to check whether the injection is successful or not (challenge **C3**).

A. Clock (in)accuracy

As mentioned earlier, the start of transmission of a *Master* frame in a given *connection event* is used as a time reference, named *anchor point*. Theoretically, given an *anchor point* t_n, the next *anchor point* should occur at t_{n+1} according to the formula 3.

$$t_{n+1} = t_n + d_{connInterval} \tag{3}$$

An attacker cannot inject a frame at this specific time, as this frame would collide with the legitimate *Master*'s packet. However, the legitimate devices involved in an established connection use multiple timers based on a clock named *Sleep Clock*. As this clock can introduce a drift in time, the *Slave* cannot assume that its *Sleep Clock* is perfectly synchronised with the *Master*'s and should listen for an extra time before and after the timing estimated from the *anchor point*.

B. Window widening

The specification introduces a concept named *window widening*, which consists in extending the listening time of a given device to compensate clocks inaccuracies. In the specific case of *Slave*'s Link Layer receiving the next *connection event*, the *window widening* w is computed using formula 4.

$$w = \frac{SCA_M + SCA_S}{1000000} \times (t_{nextAnchor} - t_{lastAnchor}) + 32\mu s \tag{4}$$

- SCA_M : *sleep clock accuracy* of *Master*'s LL (in ppm),
- SCA_S : *sleep clock accuracy* of *Slave*'s LL (in ppm),
- $t_{nextAnchor}$: predicted next *anchor point* time (in μs),
- $t_{lastAnchor}$: last observed *anchor point* time (in μs).

If the *Slave* transmits a frame for every *connection event* (i.e. slave *latency* equals to 0), the formula can be rewritten:

$$w = \frac{SCA_M + SCA_S}{1000000} \times d_{connInterval} + 32\mu s \tag{5}$$

A *Slave latency* greater than 0 increases the interval between the last observed *anchor point* and the predicted next *anchor point*, resulting in a larger window. In that case, equation 5 can be considered as the minimal *window widening*.

As a consequence, given a predicted *anchor point* t_{n+1}, the *Slave* will accept the *Master*'s packet initiating the *connection event* if it is transmitted during the *receive window* from $t_{n+1} - w$ to $t_{n+1} + w$, as illustrated in figure 4.

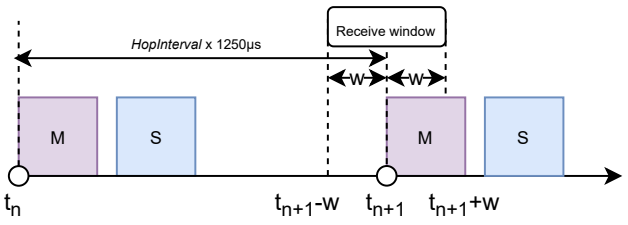


Fig. 4: *Window widening* for a *Slave* receiving the next *connection event*

C. Injecting an arbitrary packet

A frame transmitted in the previously mentioned *receive window* being considered as a *Master* packet by the *Slave*, this feature allows a race condition attack, in which an attacker can inject an arbitrary frame in an established connection by transmitting it at the beginning of the *receive window*.

For this injection to be successful, the attacker has first to be synchronised with the connection: as mentioned in the related work, multiple approaches already exist to passively sniff a connection. Second, the attacker must forge a valid frame to inject. It will be considered as new data by the *Slave* if its *Sequence Number* (denoted as SN_a) equals the *Next Expected Sequence Number* counter of the *Slave*'s Link Layer (denoted as $NESN_s$). Similarly, the *NESN* bit in the attacker frame (denoted as $NESN_a$) should indicate that the previous frame transmitted by the *Slave* (denoted as SN_s) was successfully received. Thus, the attacker should have observed in the *connection event* preceding the injection attempt a frame transmitted by the *Slave* and extracted the SN_s and $NESN_s$ bits. The SN_a and $NESN_a$ bits of the injected frame are then set according to the equation 6.

$$\begin{cases} SN_a = NESN_s \\ NESN_a = (SN_s + 1) \mod 2 \end{cases} \tag{6}$$

Third, the attacker has to calculate the *receive window* to transmit the injected frame as soon as possible during this window. He/she can use equation 5 to estimate the *window widening*. The *Master*'s *Sleep Clock Accuracy* can be extracted from the *CONNECT_REQ* packet or from control packets embedding this information (e.g. *LL_CLOCK_ACCURACY_REQ* or *LL_CLOCK_ACCURACY_RSP*). The *Slave*'s *Sleep Clock Accuracy* can be estimated at 20 ppm, which is the worst case from the attacker's perspective.

D. Checking the injection success

In order to perform various attacks requiring the injection of multiple frames, the attacker must be able to identify whether the injection of each frame is successful or not. This is not straightforward as even a successful injection does not always provoke an observable change in the behaviour of the *Slave* receiving the frame. Therefore, we need an heuristic that only relies on the observation of the parameters of the Link Layer, to indicate whether the injection is successful or not.

An injected frame is considered as valid by the *Slave* if:

- the injected frame is transmitted before the *Master*'s one during the *receive window*,
- the CRC of the injected frame equals the calculated one.

Let's consider an injection attempt with t_a the start time of the injected frame transmission (i.e., the beginning of the attack), d_a the transmission duration of the injected frame and t_m the beginning of the legitimate *Master*'s frame transmission.

An injection attempt may result in three different situations, as illustrated by figure 5:

a) the injected frame is transmitted in the *receive window* before the start of transmission of the legitimate frame ($t_a + d_a < t_m$)
b) the injected frame is transmitted in the *receive window*, but the end of the frame collides with the legitimate frame ($t_a + d_a \geq t_m$)
c) the legitimate frame is transmitted before the injected frame ($t_a \geq t_m$)

In situation a), the injection attempt is successful, because the two conditions are met. Situation b) can result in a successful injection if the collision does not corrupt the injected frame, otherwise the CRC is invalid and the injection attempt fails. Indeed, a collision might not result in a corruption when the power of the injected signal is by far superior to the power of the legitimate signal from the *Slave*'s perspective. It can also happen if the modification resulting from the superposition of two signals doesn't change the result of the heuristic used by the demodulator to demodulate the injected signal. This is possible in some configurations, depending on the phase difference between the injected and legitimate signals from the *Slave*'s perspective, along with the previously mentioned power difference. Situation c) leads to a failed injection attempt, because the first condition is not fulfilled.

Since an injection attempt may or may not be successful depending on the situation, the attacker can build an heuristic allowing him to know if a given injection was successful. This heuristic is based on the two previously mentioned conditions:

- the injected frame is transmitted before the *Master*'s one during the *receive window*: a direct observation of the legitimate packet transmitted by the *Master* is usually not possible because the attacker transmits its own injected packet at the same time. However, the *Slave*'s response can be used to infer this information indirectly. Indeed, if the injected frame was transmitted before the legitimate one, the *Slave* will consider the start of transmission of

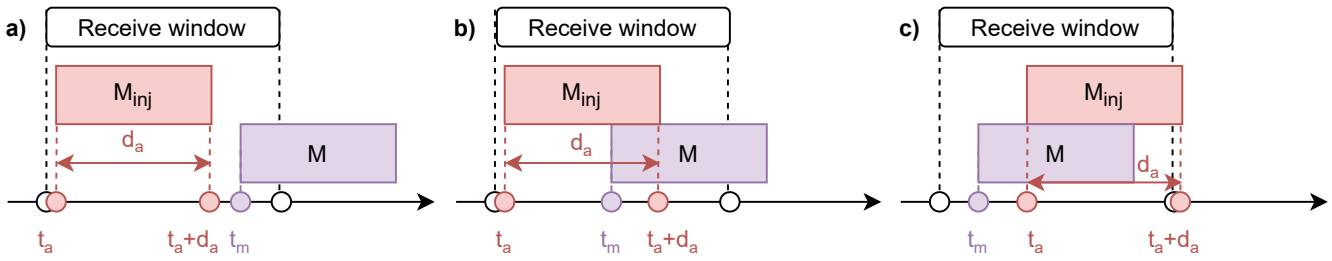


Fig. 5: Three possible outcomes of an injection attempt

the injected frame as the new *anchor point*. Consequently, the *Slave* will transmit its own frame 150 μs after the end of transmission of the injected frame. If t_s is the start of transmission of the *Slave*'s response, this requirement can be expressed as :

$$t_a + d_a + 150 - 5 < t_s < t_a + d_a + 150 + 5$$

We empirically estimated a window width of $10\mu s$, resulting in the $5\mu s$ in the above formula. This estimation has been established by injecting some specific packets that have an observable impact on the *Slave* device (e.g. transmitting a response, terminating the connection ...).

- the CRC of the injected frame equals the computed one: similarly, the attacker cannot directly check if a collision occurs and corrupts the injected frame during the transmission because he/she cannot listen to the channel during the injection. However, the *Slave*'s response can also be used to infer this information, because if the frame was received by the *Slave* with a CRC field that does not match the calculated one, the *Slave* will not change its *nextExpectedSeqNum* counter to indicate that the last received frame must be transmitted again, resulting in a *NESN* field equal to the one used in the previous frame transmitted by the *Slave*. If SN'_s is the *SN* field of *Slave*'s response and $NESN'_s$ is the *NESN* field of the *Slave*'s reponse, this requirement can be expressed as:

$$((SN_a + 1) \mod 2 = NESN'_s) \wedge (NESN_a = SN'_s)$$

Finally, the global heuristic that allows the attacker to detect the success of the injection can be expressed by the propositional formula 7:

$$\begin{aligned}(t_a + d_a + 150 - 5 < t_s < t_a + d_a + 150 + 5) \wedge \\ ((SN_a + 1) \mod 2 = NESN'_s) \wedge (NESN_a = SN'_s)\end{aligned} \quad (7)$$

with t_a the start of the transmission of the injected frame, d_a the duration of the transmission of the injected frame, t_s the start of transmission of the *Slave*'s response, SN'_s the *SN* field of the *Slave*'s response, $NESN'_s$ the *NESN* field of the *Slave*'s response.

E. Implementation

We have developed a proof of concept in order to easily perform the *InjectaBLE* attack and evaluate it. It has been implemented on a development dongle embedding a *nRF52840* chip from *Nordic Semiconductor*. This chip natively supports BLE 5.0 and allows a low level access to the *Radio peripheral*, which eases the implementation.

The dongle communicates with the *Host* using a custom USB protocol, allowing to transmit commands to the embedded software. A lightweight BLE sniffer has been implemented, based on previous works [8], [19] and [17] on BLE connection eavesdropping. When a new connection is detected by the sniffer, it synchronises with the channel hopping algorithm and transmits the received packets to the *Host*. Then, if a specific command is transmitted to the dongle, it starts the injection process and tries to inject the malicious frame defined in the command:

- before the injection, the *window widening* in use is estimated using formula 5.
- the dongle performs an injection attempt as soon as possible during the window previously defined.
- the heuristic defined in formula 7 is then used to check whether the injection was successful or not.
- if the injection attempt fails, a new one is prepared.
- if the injection attempt succeeds, a notification is transmitted to the *Host* indicating the number of injection attempts before a successful injection.

Based on this main feature, the dongle also exposes an API allowing to perform the various scenarios described in Section VI. A minimal *BLE stack* has also been implemented, to mimic the behaviour of the different roles involved in the connection.

VI. Attack Scenarios

This Section describes and illustrates four main scenarios allowing an attacker to achieve interesting offensive objectives, such as illegitimately using a device functionality, hijacking any device involved in the connection or performing a Man-in-the-Middle attack during an established connection.

A. Scenario A: illegitimately using a device functionality

This first attack scenario can be considered as the straightforward application of the injection attack. Indeed, IoT devices based on BLE usually implement the *Slave* role, so our injection approach may be used to trigger a specific functionality exposed by the targeted device. More specifically, injecting *ATT Requests* allows the attacker to interact with the *ATT* server, which is used in BLE as a generic *application layer*. Note that any *ATT* request supported by the target device could be possibly injected.

For example, an attacker could inject a *Read Request* targeting a specific handle: if the injection is successful, the *Slave* will generate and transmit a *Read Response* containing the data. It may allow him to extract interesting information from a given *characteristic*: depending on the type of device, this could have a critical impact on confidentiality. Similarly, an attacker could inject a *Write Request* or a *Write Command* to a given device. These *ATT requests* allow to modify the value of a given *characteristic*: as a consequence, the attacker is able to trigger a specific behaviour of the device, which could result in a critical impact on integrity or availability.

To illustrate the impact of this attack scenario, we have performed injection attacks targeting three commercial devices: a lightbulb, a keyfob and a smartwatch. We reverse engineered these devices to identify the type of *ATT requests* and the corresponding payloads used to trigger their main features. We then forged and injected malicious traffic triggering the following features:

- lightbulb: turning the bulb on and off, changing its colour, changing its brightness,
- keyfob: making the keyfob ring,
- smartwatch: transmitting a forged SMS to the watch.

B. Scenario B: hijacking the Slave role

This second attack scenario is aimed at hijacking the *Slave* role. If this attack succeeds, the *Slave* is forced to exit the connection, allowing the attacker to replace it without breaking the connection from the *Master*'s perspective.

This attack scenario is based on the injection of a *Link-layer control* packet: these packets are used by devices to control the connections. More specifically, the attack is based on the injection of a *LL_TERMINATE_IND* packet that is used by a device to indicate to the other one that the connection should be terminated. Since the packet injection is ignored by the *Master* and accepted by the *Slave*, it forces the *Slave* to exit the connection. However, the *Master* is not aware of the fact that the legitimate *Slave* is not present anymore: this situation allows the attacker to imitate the *Slave* behaviour in order to hijack the connection. To do so, the attacker must wait during the *inter-frame spacing* (150 μs) after the end of transmission of a *Master*'s packet before transmitting its frame, and carefully set the *SN* and *NESN* fields. This attack scenario is illustrated in figure 6.

This scenario has been successfully implemented for the three previously mentioned devices. All of them exposed a *characteristic* corresponding to the *Device Name* which allowed us to transmit a forged value "Hacked" when a *Read Request* targeting this *characteristic* was received. Let us note that such a scenario may have critical consequences depending on the type of target: as an example, an insulin pump or a pacemaker could be hijacked, allowing the attacker to transmit fake health data.

C. Scenario C and D: hijacking the Master, the Slave or both of them simultaneously (Man-in-the-Middle attack)

We have explored two other attack scenarios, based on the same approach. The scenario C consists in hijacking the *Master* role. While this kind of hijacking attack was already possible using the *BTLEJack* tool [9], its strategy is based on jamming and can easily be detected by a monitoring system. Our approach only requires the injection of a single malicious frame, making it more discrete and reliable. Scenario D allowed us to carry out a Man-in-the-Middle attack without interrupting the connection. Indeed, previous approaches to perform Man-in-the-Middle attacks [7], [15] could only be used before the initiation of the connection, which limits drastically their usability. In other words, using our strategy, an attacker could establish a Man-in-the-Middle attack **at any time**, even if a connection is already established between two legitimate devices. This strategy is critical as long term connections are very common in BLE communications, and massively used by devices such as smartwatches or trackers.

These two scenarios use a similar approach, which is based on the injection of a *CONNECTION_UPDATE PDU* as described in Section III-B: it can be used by the *Master* at any time during the connection in order to modify the parameters of the channel selection algorithm, and especially the *Hop Interval*. The attack relies on a simple idea: the attacker injects a forged *CONNECTION_UPDATE PDU* containing arbitrary parameters, indicating to the *Slave* that the connection parameters will change at a given time. When that time is reached, the *Slave* waits during the *window offset* specified by the attacker, ignoring the legitimate *Master*'s frame, then uses the new parameters while the *Master* continues to use the old ones, allowing the attacker to synchronise with the *Slave* and hijack the *Master* role or to synchronise with both of them, resulting in a Man-in-the-Middle. In the first case (e.g. *Master* hijacking) the legitimate *Master* no longer receives any response after the time at which the parameters are changed, so it leaves the connection due to timeout. Note that this approach is particularly powerful because it could also be used to hijack the *Slave* role, in a similar way to scenario B, since the attacker knows both the old and the new parameters. This approach is illustrated in figure 7.

We evaluated experimentally the *Master* hijacking using the three previously mentioned devices: with the *Master*'s role successfully hijacked, it allowed us to trigger the same features as in scenario A. Similarly, scenario D was evaluated on our three commercial devices, allowing us to arbitrarily modify the data exchanged between the legitimate devices: for example, a SMS transmitted by the smartphone to the smartwatch has been modified on the fly, or the *RGB* values describing the colour of the lightbulb have also been altered on the fly.

VII. Sensitivity analysis

We conducted several experiments to validate our attack. The objective was twofold: test its feasibility in a realistic environment and analyse the impact of different parameters upon the attack success rate. We focused on three main

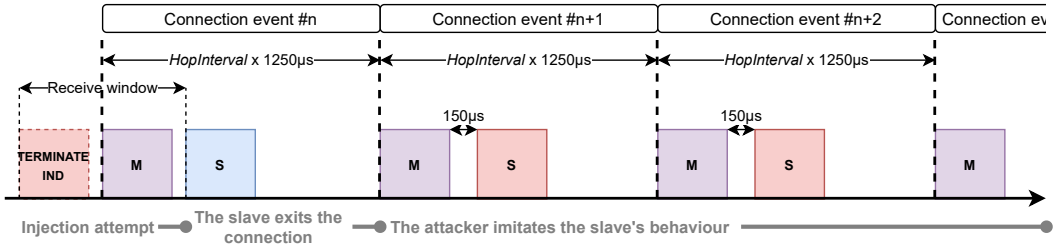


Fig. 6: Description of the slave hijacking

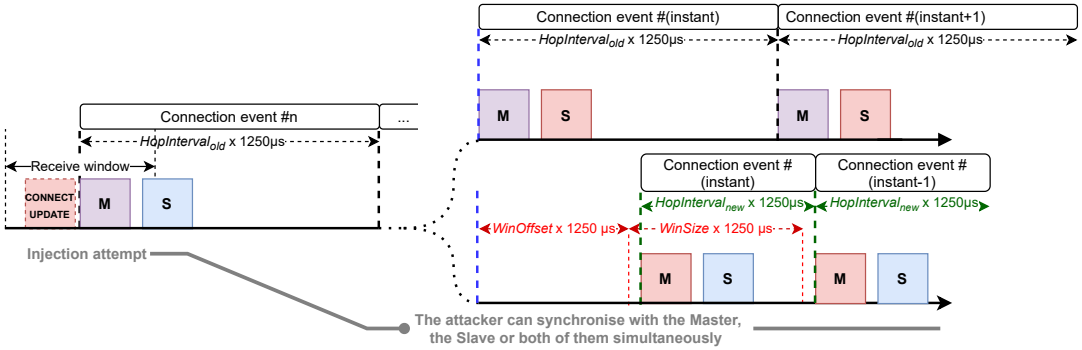


Fig. 7: Description of the Man-in-the-Middle attack

parameters that may have a significant impact on the attack success: the *Hop Interval*, the payload size and the distance between the attacker and the target *Slave*. One parameter at a time was changed and its impact on the attack success was assessed by monitoring the number of injection attempts before a successful injection.

A. Experiment 1: Hop Interval

Our first experiment focused on the *Hop Interval* parameter. Indeed, this parameter is directly involved in the estimation of the *window widening* as indicated in equation 5. Theoretically, as the attack relies on a race condition based on this window with the legitimate *Master*, the injection should be more difficult when the *Hop Interval* value is lower.

According to the specification, the theoretical *Hop Interval* range is from 6 to 3200. However, we chose to focus on six different values from 25 to 150 for two main reasons:

- We wanted to focus on the worst case of an injection attempt, which occurs when the injected frame collides with the legitimate frame, which means considering low *Hop Interval* values. Since the injected frame used during this experiment was 22 bytes long over the air (i.e., 176 μs of transmission time using the *LE 1M* physical layer), none of the *window widening* values calculated from the tested *Hop Intervals* allowed an injected frame to be entirely transmitted without a collision.
- We wanted to conduct our experiment on target real-life devices and most of them do not allow the use of high *Hop Interval* values, because the resulting connections could be extremely unstable and break quickly. We thus used the *Hop Interval* values in the range supported by a connected lightbulb, which was the commercial device supporting the widest range of *Hop Interval* values we were able to find.

To be able to precisely tune the *Hop Interval* parameter, we used a modified version of the open-source Mirage framework [11], [12] to simulate a *Central* device, because of its capability to access the *HCI* on a low level.

We reversed the communication protocol built over *GATT* used by this lightbulb, then selected a *Write Request* allowing to turn the light off as our injection frame. The corresponding payload is 14 bytes long, making the entire frame 22 bytes long. We chose a frame with a visible effect on the device to validate our heuristic.

The experimental setup was quite simple: the legitimate *Peripheral* and *Central* devices and the attacker were placed on the three vertices of an equilateral triangle, with 2 meters edges. The *Central* initiates connections with the *Peripheral* repeatedly while the attacker synchronises with these connections and starts the injection attack at a specific *connection event*. The experiment was conducted in a realistic environment, including several other BLE devices and multiple *WiFi* routers. Let us note that synchronising the attack tool with a connection is not trivial, especially in such a noisy environment. For each *Hop Interval* value, we performed 25 injection attacks, and monitored the number of injection attempts required before a successful injection. The results are presented in figure 9.

The attack was successful for every tested connection. The variance of the number of unsuccessful attempts decreases

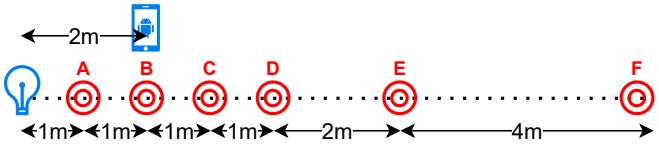


Fig. 8: Experimental setup

quickly between 25 and 100, and stabilises afterwards. Similarly, the median value remains at a low value less than 4. These results show that the injection is always feasible even with small *Hop Intervals*, and the number of injection attempts required before a successful injection is generally low. The experiment confirms that the *Hop Interval* has a significant impact on the injection attack success. However, the injection is more reliable with higher values.

B. Experiment 2: Payload size

This experiment was focused on the payload size of the injected frame, and was intended to empirically confirm that injecting shorter frames increases the probability of success.

The experimental setup and the environment are similar to the one presented above. We selected four different values of payload size: 4, 9, 14 and 16, which correspond to frames that have an observable effect on the target lightbulb (such as disconnecting it, turning it off, or changing its colour), allowing to confirm the success of an injection attempt independently from our success detection heuristic.

We repeated the experiment 1, this time with a fixed *Hop Interval* of 75, and iterating over the different payload sizes. The results are displayed in figure 9.

Similarly to experiment 1, we observe higher reliability when the payload size decreases, which is consistent with the theory as a smaller portion of the injected frame collides. The number of injection attempts required before a successful injection remains very low (less than 3 for the median).

C. Experiment 3: distance

Our last experiment was conducted to evaluate the impact of the distance between the attacker and the legitimate *Peripheral*. Theoretically, since the distance impacts the signal strength of the injection from the *Peripheral*'s perspective, it may lower even more the success rate when a collision with the legitimate frame occurs. We used the same lightbulb as *Peripheral*, but used a smartphone as legitimate *Central* to get closer to a real-life scenario. The phone was used to establish 25 connections per tested distance, using its default *Hop Interval* value equal to 36. As we chose to only inject the 22 bytes long *Write Request* allowing to turn the bulb off, this *Hop Interval* value doesn't allow for collision-free transmissions.

The experimental setup was slightly different from the one used in experiments 1 and 2: we placed the lightbulb and the phone within two meters of each other, then we tested six different positions for the attacker, from 1 to 10 meters, as illustrated in figure 8. This allowed us to evaluate the attack success when the attacker is closer to the *Peripheral* than the legitimate *Central* (position A), when they are at the same distance (position B) and when the attacker is further (positions C,D,E and F).

The results are presented in figure 9. They show a significant impact of the distance between the attacker and the *Peripheral* on the reliability, as the variance increases when the distance is higher. It validates our assumption that the attacker has an higher probability to quickly perform a successful injection if closer to the target. However, let us note that each tested connection leads to a successful injection: it means that the attacker can perform a successful attack from every location, including position F which is 10 meters away from the *Peripheral*, while the legitimate *Master* is only 2 meters away. This experiment highlights the practical feasibility of the attack, and shows that, even under adverse conditions in a realistic environment, the attack is still possible.

We also tested the attack effectiveness behind a wall, to evaluate the impact of obstacles. The experimental setup was very similar to the distance experiment: the lightbulb (legitimate *Peripheral*) and the phone (legitimate *Master*) were placed within two meters of each other in the same room, while the attacker was located at four different positions behind a wall, from 2 to 8 meters from the *Peripheral*. Similarly to the other experiments, we established 25 connections per tested distance and measured the number of injection attempts needed before a successful injection. Results are presented in figure 9: as expected, the presence of a wall increases the number of injection attempts needed to perform a successful injection, and the variance increases with the distance. However, even if the attack requires more attempts, we managed to successfully inject a frame for every connection we tested, even in the worst case from the attacker's perspective. These results show that this attack is realistic and could be carried out even if the attacker is not in the same room as the target.

VIII. Counter-measures

The *InjectaBLE* attack exploits a vulnerability that is inherent to the BLE protocol specification. As a result, we should consider every BLE communication as potentially vulnerable and the environments exposed to BLE devices should be designed and monitored with the assumption that some attacks could potentially be carried out through legitimate communications. Several counter-measures could be investigated either to limit the impact of the attack, or to prevent or detect it.

As explained in section III-B, the practical implementation of the *InjectaBLE* attack requires injecting arbitrary frames at specific moments. Three solutions could be investigated. Each one requires more or less deep changes in the BLE stack or in the usage of BLE chips. These changes may not be appropriate from the user's point of view in the case of an industrial environment, because of a possibly high number of devices to reprogram and the cost of certification processes.

The first solution deals with some communication time parameters of the stack itself. For example, by reducing the duration of the widening windows the possibility for an attacker to inject a frame at the right time will be mechanically reduced.

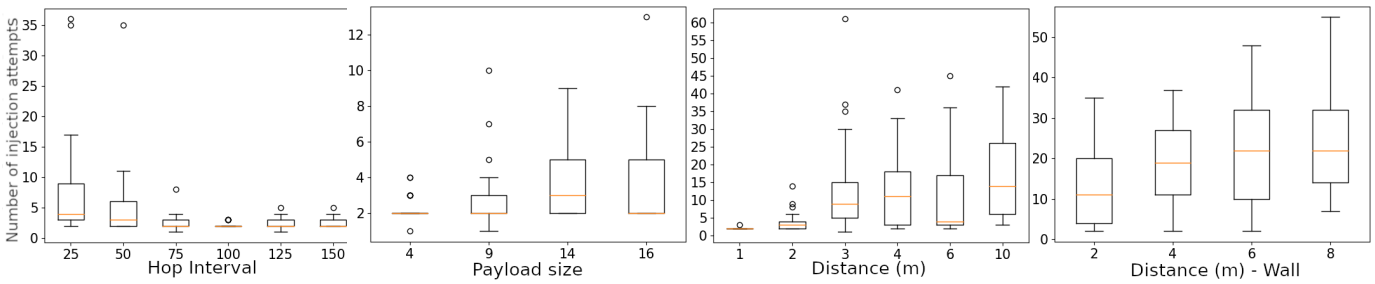


Fig. 9: Experiment Results

More precisely, the rate of successful injection will decrease due to the collision with a legitimate frame. However it should be noted that such an approach requires changes to the BLE standard which could have side effects on the reliability and stability of the communications. The second solution is slightly less restrictive. Without going as far as modifying the BLE standard, it requires to systematically activate the encryption mechanisms defined in BLE specification. If all frames are correctly ciphered, an attacker will not be able to easily sniff the connection parameters and forge a valid frame. In this specific case, the vulnerability is still present, even if its impact is limited to Denial of Service attacks. While this solution could be straightforward, it is not in reality. It must be noted that the majority of BLE communications are poorly or not at all encrypted today (see [26] for a qualitative study of the percentage of BLE devices activating encryption mechanisms). As a consequence, in most cases, this countermeasure requires end-users to reprogram all their devices, which could be tricky, especially in the context of industrial devices.

During our experiments, we noticed that some manufacturers do not use the native protocol encryption but rather choose to implement their own over the *GATT* application layer. We strongly advise against this solution, since in this case the *LL control frames* will not be encrypted and we have already demonstrated in our attack scenarios that an attacker could achieve interesting objectives by injecting this kind of frames, such as initiating a Man-in-the-Middle and not forwarding the legitimate traffic to perform a denial of service.

The last solution is based on a non intrusive approach. Defensive solutions dedicated to the IoT context could be considered to monitor and detect in real time or not attacks targeting wireless protocols. An *Intrusion Detection System* (*IDS*) designed to monitor BLE Link Layer could be able to detect, at the right instant, the presence of double frames: the legitimate *Master* frame and the attacker one. For instance, the *IDS* proposed in [18] is able to identify deviations from legitimate behaviour by monitoring the radio activity of the wireless environment. F. Galtier et al. also propose, in [13], an *IDS* able to fingerprint legitimate devices (based on physical characteristics of the radio signals) and to detect inappropriate fingerprints related to the attacker frames. Monitoring solutions designed to detect BLE spoofing attacks, such as [24] or [25], may also detect behavioural anomalies in the communication between the devices, for example variations in the timing between packet emissions or change of BLE profile, and hence detect the injection attempts. Machine Learning oriented solutions can also be effective, as in [16], where A. Lahmadi et al. used Neural Networks to build an attacker model, and detect Man-in-The-Middle attacks.

IX. Conclusion

In this paper, we demonstrated the feasibility of a new injection attack named *InjectaBLE* targeting the BLE protocol, allowing to inject malicious frames into an established connection. This attack significantly increases the attack surface of BLE communications, because it exploits a vulnerability of the specification itself independently of the stack implementations, and can be achieved quite easily using common BLE chips. We analysed the impact of multiple factors on the attack success rate and demonstrated that exploiting this weakness could allow an attacker to perform critical attack scenarios that were not realistic until now, such as *Slave* hijacking or *Man-in-the-Middle* attack targeting established connections. We also performed sensitivity analyses that showed that this injection always succeeds in various experimental conditions.

Activating the BLE native cryptographic mechanisms can efficiently mitigate this attack. However, in practice, the vast majority of commercial devices do not use encryption, making them vulnerable by design to *InjectaBLE*. The results presented in this paper clearly highlight the need to generalise the systematic use of encryption in BLE communications.

The new offensive capabilities provided by *InjectaBLE* open opportunities for other critical attack scenarios that need to be carefully investigated. For example, being able to hijack the *Slave* role may potentially allow an attacker to transmit an ATT notification indicating that the *ATT* server structure has been modified: it could be used to expose a malicious keyboard profile instead of the original one, and inject keystrokes to the *Master* by implementing *HID over GATT* protocol. We plan to explore the practical feasibility of this attack as future work. In parallel, it's important to design efficient defensive approaches, e.g., a passive monitoring system allowing to detect *InjectaBLE* in real time.

References

[1] D. Antonioli, N. O. Tippenhauer, and K. Rasmussen, "Key negotiation downgrade attacks on bluetooth and bluetooth low energy," *ACM*

Trans. Priv. Secur., vol. 23, no. 3, Jun. 2020. [Online]. Available: https://doi.org/10.1145/3394497

[2] D. Antonioli, N. O. Tippenhauer, K. Rasmussen, and M. Payer, "Blurtooth: Exploiting cross-transport key derivation in bluetooth classic and bluetooth low energy," 2020.

[3] Armis, "Blueborne Technical White Paper," https://go.armis.com/hubfs/BlueBorne Technical White Paper.pdf, 2017.

[4] ——, "BleedingBit Technical White Paper," https://go.armis.com/hubfs/BLEEDINGBIT - Technical White Paper.pdf, 2018.

[5] *Bluetooth Core Specification*, Bluetooth SIG, 12 2019.

[6] S. Bräuer, A. Zubow, S. Zehl, M. Roshandel, and S. Mashhadi-Sohi, "On practical selective jamming of bluetooth low energy advertising," in *2016 IEEE Conference on Standards for Communications and Networking (CSCN)*, 2016, pp. 1–6.

[7] D. Cauquil, "BtleJuice, un framework d'interception pour le Bluetooth Low Energy," https://www.slideshare.net/NetSecureDay/nsd16-btlejuice-un-framework-dinterception-pour-le-bluetooth-low-energy-damien-cauquil, 2017.

[8] ——, "Sniffing btle with the micro:bit," *PoC or GTFO*, vol. 17, pp. 13–20, 2017.

[9] ——, "You'd better secure your BLE devices or we'll kick your butts !" 2018, https://media.defcon.org/DEF CON 26/DEF CON 26 presentations/Damien Cauquil - Updated/DEFCON-26-Damien-Cauquil-Extras/.

[10] ——, "Defeating Bluetooth Low Energy 5 PRNG for fun and jamming," 2019, https://media.defcon.org/DEF CON 27/DEF CON 27 presentations/DEFCON-27-Damien-Cauquil-Defeating-Bluetooth-Low-Energy-5-PRNG-for-fun-and-jamming.PDF.

[11] R. Cayre, "Mirage github repository," https://github.com/RCayre/mirage/.

[12] R. Cayre, V. Nicomette, G. Auriol, E. Alata, M. Kaâniche, and G. Marconato, "Mirage: towards a metasploit-like framework for iot," in *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2019, pp. 261–270.

[13] F. Galtier, R. Cayre, G. Auriol, M. Kaâniche, and V. Nicomette, "A psd-based fingerprinting approach to detect iot device spoofing," in *25th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC 2020)*.

[14] M. E. Garbelini, C. Wang, S. Chattopadhyay, S. Sumei, and E. Kurniawan, "Sweyntooth: Unleashing mayhem over bluetooth low energy," in *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. USENIX Association, Jul. 2020, pp. 911–925. [Online]. Available: https://www.usenix.org/conference/atc20/presentation/garbelini

[15] S. Jasek, "Gattacking Bluetooth Smart Devices," 2017.

[16] A. Lahmadi, A. Duque, N. Heraief, and J. Francq, "Mitm attack detection in ble networks using reconstruction and classification machine learning techniques," in *MLCS 2020-2nd Workshop on Machine Learning for Cybersecurity*, 2020.

[17] S. Qasim Khan, "Sniffle: A sniffer for Bluetooth 5 (LE)," 2019, https://hardwear.io/netherlands-2019/presentation/sniffle-talk-hardwear-io-nl-2019.pdf.

[18] J. Roux, E. Alata, G. Auriol, M. Kaâniche, V. Nicomette, and R. Cayre, "Radiot: Radio communications intrusion detection for iot-a protocol independent approach," in *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)*. IEEE, 2018, pp. 1–8.

[19] M. Ryan, "Bluetooth: With Low Energy comes Low Security," 2013.

[20] ——, "How Smart is Bluetooth Smart ?" 2013.

[21] A. Santos, J. Filho, A. Silva, V. Nigam, and I. Fonseca, "Ble injection-free attack: a novel attack on bluetooth low energy devices," *Journal of Ambient Intelligence and Humanized Computing*, 09 2019.

[22] M. von Tschirschnitz, L. Peuckert, F. Franzen, and J. Grossklags, "Method confusion attack on bluetooth pairing," in *2021 2021 IEEE Symposium on Security and Privacy (SP)*. Los Alamitos, CA, USA: IEEE Computer Society, may 2021, pp. 213–228. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SP40001.2021.00013

[23] J.-L. Wu, Y. Nan, V. Kumar, D. Tian, A. Bianchi, M. Payer, and D. Xu, "Blesa: Spoofing attacks against reconnections in bluetooth low energy," in *WOOT @ USENIX Security Symposium*, 2020.

[24] J. Wu, Y. Nan, V. Kumar, M. Payer, and D. Xu, "Blueshield: Detecting spoofing attacks in bluetooth low energy networks," in *23rd International Symposium on Research in Attacks, Intrusions and Defenses ({RAID} 2020)*, 2020, pp. 397–411.

[25] M. Yaseen, W. Iqbal, I. Rashid, H. Abbas, M. Mohsin, K. Saleem, and Y. A. Bangash, "Marc: A novel framework for detecting mitm attacks in ehealthcare ble systems," *Journal of Medical Systems*, vol. 43, no. 11, p. 324, 2019.

[26] C. Zuo, H. Wen, Z. Lin, and Y. Zhang, "Automatic fingerprinting of vulnerable ble iot devices with static uuids from mobile apps," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 1469–1483.