CAMEL: Mitigating Bluetooth Low Energy Packet Injection Attacks

Benoit Knuchel[1,2], Damian Vizár[1],
Chitchanok Chuengsatiansup[3,4], and Slim Fatnassi[1]

[1] CSEM, Neuchâtel, Switzerland
{benoit.knuchel, damian.vizar, slim.fatnassi}@csem.ch
[2] AAU Klagenfurt, Klagenfurt, Austria
[3] Hasso Plattner Institute, Potsdam, Germany
{chitchanok.chuengsatiansup}@hpi.de
[4] University of Potsdam, Potsdam, Germany

Abstract. Bluetooth Low Energy (BLE) is the prevailing low-power wireless communication protocol for Internet of Things devices (IoT). While the protocol security layer is sound, recent research revealed a number of implementation-related vulnerabilities. For example, a race condition vulnerability can be exploited to inject malicious packets, enabling Man-in-the-Middle (MitM) attacks, device hijacking, or efficient Denial-of-Service (DoS) attacks against millions of IoT devices that implement no BLE encryption even today. Recognizing the possible impact, researches proposed detection method for BLE packet injection in the OASIS paper. We experimentally demonstrate that OASIS' state-of-the-art BLE packet injection detection can be reliably bypassed for $\approx 97.7\%$ of the possible values of the BLE connection parameter called the hop interval. We then propose CAMEL, a lightweight, adaptive mitigation that secures a much broader configuration space without sacrificing resource efficiency. CAMEL employs a host-based, probabilistic detection mechanism that establishes a detection threshold from the observed timings of packet transmissions across successive connection events, optimally balancing false positive rates against the likelihood of a successful attack for each given connection. Hardened with our probabilistic recalibration, our method represents a practical and resource effective detection of injection attacks that can help prevent MitM and other serious attacks on millions of IoT devices without BLE encryption.

1 Introduction

Bluetooth Low Energy (BLE) is a short-range, wireless communication protocol known for its eponymous low energy consumption and a widespread deployment across industries. With more than 5 billion BLE devices shipped in 2024 alone [1], a single exploitable vulnerability can compromise entire fleets of wearables, headsets, smart home appliances, industrial IoT devices, asset trackers (e.g., Apple's iBeacon or Googles Eddystone) and so on. One such vulnerability is proneness to packet injection attacks, which can enable man-in-the-middle (MitM) attacks,

device hijacking, or terminate connections with a single packet—impacting, both consumer and industrial deployments. The vulnerability is linked to BLE's strategy to achieve the low energy consumption. BLE devices sleep most of the time and communicate in short bursts, at a pre-arranged time interval. The tight synchronization of the devices' clocks needed for the coordinated wake-ups is never perfect in practice. To compensate for the inevitable clock drift, the receiving device accepts frames in a so-called *widening window*, centered around the theoretical reception time, called the *anchor time*. This margin gives rise to a race condition, allowing attackers to exploit the widening window for a malicious packet injection.

The most notable attack exploiting this margin, InjectaBLE [2], leverages the window-widening tolerance in the Link Layer (LL): by injecting a valid LL Protocol Data Unit (PDU) earlier than the legitimate transmission within the receive window, the receiver decodes the attacker's frame first. The attacker estimates anchor timings across successive events to schedules the injection; with 1-2MBit/s data rate, the airtime of short LL control PDUs (tens of μs) is well within the typical window, making the injection of malicious PDUs without interference practical. Using the Mirage framework [3] on commodity hardware, Cayre et al. implemented concrete attacks on live connections, including role hijacking of either peripheral or central, Man-in-the-Middle by interposing attacker traffic, or connection termination (DoS) via link-layer control frames. These attacks matter beyond the lab. The most impactful attacks powered by InjectaBLE (hijacking and MitM) only apply to connections that do not use link encryption. Yet even today, various IoT devices [4–6] do not implement any security at all, with a recent survey [7] suggesting that only less than 5% of BLE devices implemented the secure pairing methods. When combined with the number of shipped BLE chips, this represents an astronomical number of devices vulnerable to these attacks. Even with connection security, standard or otherwise, an efficient DoS attack is possible through a repeated connection drop by an early injection of LL control frames, potentially turning compromised IoT devices into long-lived BLE DoS platforms.

OASIS [8] is a landmark, portable embedded intrusion detection system (IDS) for BLE controllers, consisting of five low-level detection modules. One of these is a dedicated packet-injection detector, which flags frames that arrive "too early" in the window via an empirically determined threshold, underpinning the system's MitM assurances. We show that the effectivity of this threshold is highly sensitive to the *hop interval* parameter; for $\approx 97.7\%$ of the possible hop interval values (and $\approx 92.9\%$ of the empirically tested hop interval values), attackers retain a near-100% success rate while remaining undetected. Our experimental invalidation of packet injection detector, an integral part of OASIS, across a broad configuration space is security-significant: it reinstates practical feasibility of spoof/MitM/DoS on fleets previously thought protected. This gap motivates a more adaptive but resource-efficient countermeasure.

Our objective is a defense that is practically deployable across commodity IoT platforms, which frequently operate under modest flash/RAM budgets and

strict energy constraints. In many consumer and edge deployments, the per-device stakes of an isolated breach are limited, whereas the primary risk materializes at scale, as low-cost nodes are co-opted in bulk (e.g., Mirai [9]) and externalities dominate the impact [10]. Consequently, the most relevant attacks to deter are those feasible with unmodified commodity hardware: for instance, widely available BLE dongles such as the nRF52840 used in prior active attacks, and, by extension, actions sourced from already-compromised IoT nodes. While high-power or specialized RF adversaries are conceivable, a low-overhead technique that significantly raises the cost of low-cost, scalable attacks should not be dismissed: when aligned with baseline guidance and market incentives, it can be a cost-effective deterrent for the threat most likely to impact fleets.

1.1 Our Contributions

Our contribution is twofold.

1. We analyze the packet injection detector from OASIS, identify a serious configuration-dependent blind spot allowing it to be bypassed in a majority of possible connection configurations and experimentally confirm that a successful, undetected attack is possible with practically used connection hop interval values (Section 2.4).
2. We introduce **C**ollision prob**A**bility-optimizing **M**itigation of packet inj**E**ction attacks against b**L**e, an improved detection method, which sets a tighter early frame rejection threshold, which is crucially determined dynamically, from live packet timings, thus accounting for variations in packet arrival time distribution across different devices and communication environments (Section 6.2).

Inspired by our experimental analysis of the temporal distribution of legitimate packet arrival times with several (Section 5), our approach is based on a modelling of the probability distribution of legitimate packet arrival times within the widening window. Our threshold computation method redefines the lower-threshold of the widening window using the initial observed packets arrival distribution. By carefully defining and live-tuning detection thresholds, we achieve a balance between false positive rates and the probability of successful packet injection attacks. We analyze the security afforded by CAMEL in a variety of adversarial scenarios.

Crucially, CAMEL requires no per-device, per-stack, or per-link manual configuration: the threshold is inferred online from the first live timings, ensuring portability across heterogeneous radio chips and partner devices without any retuning. This is critical for adoption and practical applicability, eliminating the otherwise necessary cross-product validation of each controller with likely peers to confirm a safe configuration.

1.2 Related Work

A number of works have reported a wide variety of real-world exploitable BLE vulnerabilities such as software bugs (SweynTooth [11]), protocol vulnerabilities

(Mirage [3], BLESA [12], InjectaBLE [2]), cryptographic flaws (SweynTooth), etc. Others then proposed countermeasures against these vulnerabilities, notably in BlueShield [13], MARC [14], RadIoT [15], and more [16]. Among these, two, notable recent results are OASIS [8] (discussed above), and VaktBLE [17]. OASIS is a host-based BLE IDS, embedded directly inside BLE controller firmware to detect and mitigate packet injection (InjectaBLE), jamming/DoS, MitM/hijacking, and security downgrades (e.g., KNOB)—by instrumenting link-layer events and timing within the controller, yet not all as effective as initially believed. VaktBLE is a network-based design. While it has an excellent coverage against numerous BLE attacks, it requires a dedicated monitoring device acting as a transparent packet monitor. This limits the applicability in most use cases, and in particular makes a retrofit at scale through a firmware update impossible.

2 Background

2.1 Connection Events

In BLE, the two communicating devices take the roles of so-called *central*, which scans for advertising peripherals and initiates connection, and *peripheral*, which advertises and awaits for a central to connect. When a connection is established, the central and the peripheral synchronize clocks and agree on a periodic time interval, known as the connection interval, *connInterval* in the Bluetooth Specifications [18]. At each connection interval, the transmission of data occurs during the so-called connection events. In the i^{th} connection event, central transmits a packet p_i, and peripheral responds with another packet. Even if there is no data to transmit, the connection is maintained by sending an empty packet. The precise time at which the central device begins its i^{th} transmission (and the peripheral its i^{th} reception) is referred to as the *anchor point* t_i, such that $connInterval = t_{i+1} - t_i$. In reality, the two devices will be unable to perfectly synchronize on each anchor point, as various sources will contribute to a clock drift. To compensate, the protocol allows for transmission to occur in a short interval around the anchor point called the *widening window*, shown in Fig. 1. The next packet, p_{i+1}, has to arrive in the interval $[t_{i+1} - w, t_{i+1} + w]$, with t_{i+1} the next anchor point and w half the size of the widening window.

From the Bluetooth Specifications [18], the connection interval must be a multiple of $1.25ms$ within the range from $7.5ms$ to $4s$. It is defined as $connInterval = n \cdot 1.25$ with $n \in [6, 3200]$. This parameter n is part of the connection parameters established during connection setup. The formula to compute the widening window can also be found in the Bluetooth Specifications [18] Volume 6, Part B, Section 4.2.4. It depends mainly on the parameter n defined above and the Sleep Clock Accuracy (SCA) of both the peripheral and the central devices.

2.2 InjectaBLE - Race Condition Vulnerability

By accepting any transmission occurring within the widening window interval, the protocol introduces a race condition vulnerability named InjectaBLE [2]. In

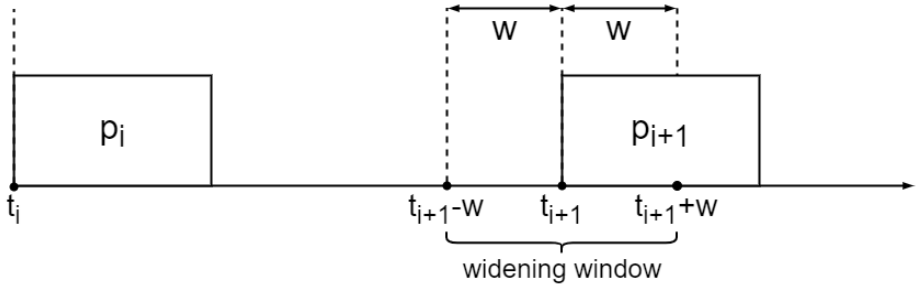

Fig. 1. BLE widening window.

an unencrypted connection, an attacker that is able to inject an entire packet after the start of the widening window but before the legitimate device's packet arrives, can spoof the legitimate device or even hijack the connection using specialized control packets. Such a control packet could be, for example, `LL_TERMINATE_IND`, to terminate the connection, which would allow the attacker to then connect to the central, impersonating the peripheral. There are three distinct cases for the effects of this packet injection attack (see Fig. 2):

1. **No Collision:** The entire malicious packet is injected before the arrival of the genuine packet. The attack is successful.
2. **Injected packet arrives first, collision occurs:** The transmission of the malicious packet starts before the legitimate one but the two transmissions partially overlap, which causes a collision between the two packets. In our model, this always leads to a mutual jamming of the two packets. Outside of the threat model, it is hypothetically possible for either the attacker packet (e.g., if the attacker transmits with much more power and the receiver radio has an adaptive gain) or legitimate packet (e.g., with a reverse transmission power ratio and collision early in the preamble) to be received correctly. A more detailed discussion of attacker success in face of collisions is given in Section 7.1.
3. **Injected Packet Arrives Late:** The transmission of the malicious packet starts after the legitimate one, resulting in a failed attack every time in the model.

2.3 BLE Control Packet Timing

As per the Bluetooth specifications [18], Volume 6, Part B, Section 2.1, the smallest possible packet has a size of 10 bytes with $1Mbps$ transmission rate or 11 bytes with the $2Mbps$ transmission rate. Specifically, the minimum size of the PDU is 2 bytes. From Section 2.4.2, the `LL_TERMINATE_IND` control packet's PDU is also 2 bytes, making it of the smallest possible size. Such a packet can thus be transmitted within $44\mu s$ at $2Mbps$, or $80\mu s$ at $1Mbps$. This is important because the longer it takes to transmit the malicious packet, the longer it is in

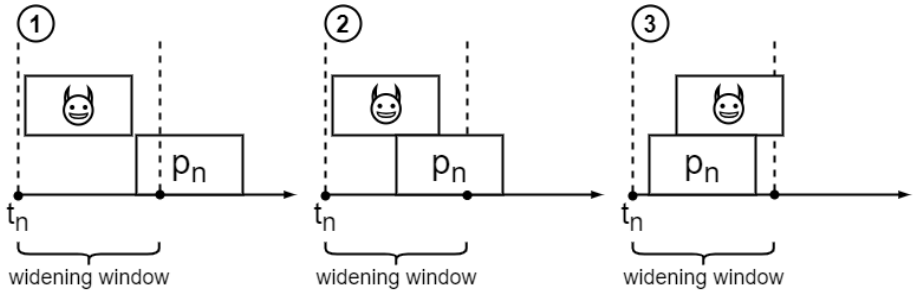

Fig. 2. Scenarios of injected malicious packets in the widening window.

transmission, and so the more difficult it is for the attacker to transmit it without causing a collision with the legitimate one and hence fail.

2.4 OASIS Solution

Observing that genuine packets tend to arrive later than malicious ones within the widening window, OASIS [8] implemented a countermeasure consisting of the flagging of all packets arriving before a certain threshold in the widening window. The threshold is derived dynamically from the last k observed legitimate packets as the median arrival time (relative to the anchor point) of the last k legitimate packets minus a margin of error.

This approach has one notable limitation: the margin of error is a fixed multiple of the observed median. This fails to account for variations in parameters such as the connection interval, the hardware used, and other dynamic conditions. Another problem is that the median does not capture the spread of legitimate packet arrival times, making it difficult to assess how the threshold affects collision rates and false positive rates, let alone find a configuration that would effectively protect in the variety of possible conditions. For instance, if the legitimate packet arrival times distribution has a big spread and a high variance, the threshold could lead to a high false positive rate.

Addressing the InjectaBLE vulnerability has not been the sole focus of the OASIS paper. The authors tackled multiple BLE vulnerabilities and InjectaBLE did not get a lot of attention. In particular, not much indication is given on how the error margin was determined, only that it has been done empirically. [5] The presented concept of reducing the widening window by filtering out packets arriving before a dynamic threshold is promising, but a comprehensive analysis is lacking. As a result, the countermeasure is not always effective.

[5] We attempted to clarify this in personal correspondence but haven't gotten a reply.

3 Threat Model

We consider an attacker within radio range, capable of sending arbitrary packets. In our model, the time of legitimate packet reception starts according to an underlying probability distribution specific to the given pair of devices and connection parameters selected, and the attacker is given the power to arbitrarily control the timing of malicious packet transmission. We abstract away various environmental factors, such as physical distance and obstacles, the effects of varying transmission power of the attacker and the legitimate device, as well as the possibly varying effect of packet collision occurring in different packet parts, e.g. collision starting in the first packet's preamble as opposed to its payload. Specifically, we assume that any collision in the transmission of two packets prevents either of them from being received correctly. This assumption is reasonable, as achieving successful packet injection despite a collision requires specific, favorable conditions to be arranged by the attacker, which is why we leave it out of scope of this paper. Further discussion is given in Section 7.1.

4 Analysis of OASIS

4.1 Alert Condition

Packet injection countermeasures from OASIS are ineffective in many cases, depending on the connection parameters. Section 4.3 presents its theoretical limitations. From the code available on GitHub [19], we found that the condition to detect a malicious packet is defined as follows:

$$|t_{act,i+1} - median_i| \,/ median_i \geq 0.0005 \tag{1}$$

Here, $t_{act,i+1}$ is the time elapsed between the start of the $(i+1)^{th}$ and the i^{th} packet and $median_i = \text{med}(t_{act,i-23}, \ldots, t_{act,i})$, the median of the last 24 observed connection intervals. As an example, if every packet arrived precisely at the anchor time, we would have $median_i = t_{act,i} = t_{i+1} - t_i$ from Fig. 1.

Note that this is an interpretation of the available code. We are not sure it was the intended solution, but we took the tightest possible bound presented. See Appendix A.1 for further explanations.

4.2 Bypassing the Detection

For the attacker to succeed, the timing of the malicious packet must respect the condition from Eq. 1, i.e. $t_{act,i+1} - median_i < median_i \times 0.0005$ in order to not be detected. Furthermore, it should also avoid colliding with the legitimate one, which can be expressed as:

$$median_i - t_{act,i+1} \geq 44\,\mu s \tag{2}$$

Using the median in this inequality makes it easier to define, but it gives some uncertainty around the arrival time of the legitimate packet. If an attacker

sends the malicious packet such that it arrives precisely at the threshold, under condition in Eq. 2, it still has a 50% chance of colliding with a legitimate packet and hence fail. In practice, legitimate packets may arrive significantly after the median, reinforcing the need for a more conservative estimate.

4.3 Theoretical Limitations

Assuming the tightest constraint for the attacker, i.e. $median_i - t_{act,i+1} = 44$, the condition to pass through OASIS detection is:

$$44 \geq median_i \times 0.0005 \tag{3}$$

We also know that in theory the median of the last k received packets is the $connInterval$. From section 2.1, we can find that $connInterval = n \cdot 1250\mu s$, which can substitute $median_i$, and we get:

$$44 \geq n \times 1250 \times 0.0005 \tag{4}$$

$$n \geq 71 \tag{5}$$

So, when $n \geq 71$, meaning that when $connInterval \geq 88.75$ms, the attacker has sufficient time to inject a packet without causing a collision or triggering the OASIS detection system. Note that this condition on the connection interval is in the low-end of its possible values since it can go up to $4s$.

4.4 Practical Verification

To validate this analysis, we established a BLE connection between an ESP32-WROVER-IE as the central and an nRF52840-DK as the peripheral. For both devices, we used the Apache Mynewt NimBLE BLE stack. To launch the attack, we used an nRF52840 Dongle, flashed with the InjectaBLE firmware [20], and using the Mirage [3] framework. We set up the connection with $n = 144$ and a transmission rate of $1Mbps$. Under these conditions, OASIS will accept any packets arriving after the threshold set at $median - 90\mu s$. With this setup, we get a widening window of $198\mu s$ (computed by the BLE stack), and the `LL_TERMINATE_IND` requires $80\mu s$ to be transmitted.

To illustrate this, Fig. 3 shows the arrival time of 2500 legitimate packets. We can see that between the OASIS threshold at $-90\mu s$ and the earliest genuine traffic at $-4\mu s$, there is enough time for an attacker to send the desired packet taking $80\mu s$. In our live experiment, the attack went undetected.

5 Legitimate Traffic Analysis

Similar to CAMEL, implicit to the OASIS packet injection detector is the assumption that the initial traffic after the connection establishment is free of attacker influence to dynamically establish a detection threshold adapted to the

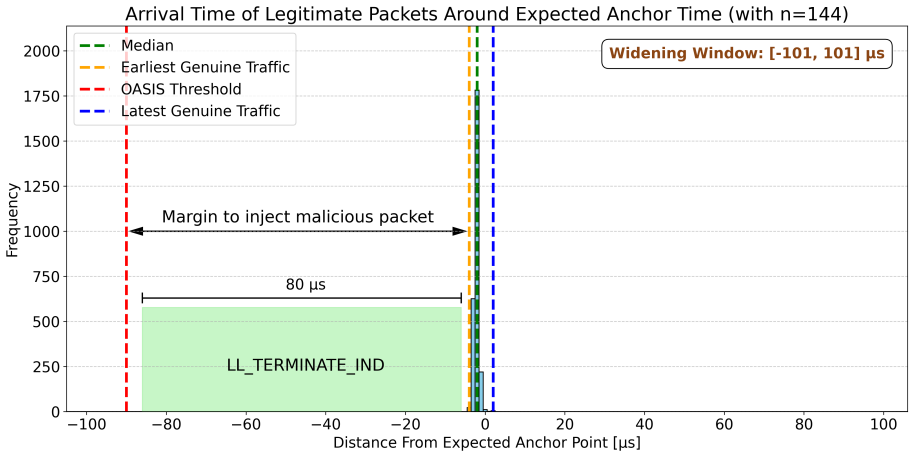

Fig. 3. Practical verification of OASIS limitations: a packet bypassing the detection. The green rectangle represents the reception of a maliciously injected packet inside the margin left between the threshold set by OASIS and the earliest arrival time of genuine packets (so that no collision happens). `LL_TERMINATE_IND` is a link layer packet sent to end a connection.

legitimate communication patterns. Both solutions effectively assume that the arrival times of legitimate packets are within a cluster that is much tighter than the full widening window, bar a few possible outliers. To validate this assumption experimentally, we measured the arrival time of the first bit of packets received by the peripheral from the central. We conducted this experiment under different conditions:

1. Using the same pair of hardware *model* but with different instances, i.e. multiple units of nRF52840-DK and ESP32-WROVER-IE.
2. Using different hardware models, i.e. replacing an ESP32-WROVER-IE with an iPhone13.
3. Using the same hardware but with different connection parameters, i.e. modifying the connection interval.

We used these off-the-shelf devices: nRF52840-DK, ESP32-WROVER-IE, and an iPhone13. The nRF52840-DK runs the Apache Mynewt RTOS with the NimBLE BLE stack. It acts as the peripheral. The central is either an ESP32-WROVER-IE running FreeRTOS with Nimble BLE stack, or an iPhone 13.

In BLE, the peripheral computes the next packet's anchor time by timestamping the arrival of the current packet's first bit and adding the connection interval. This timestamping is inherently part of the controller's operations.

By analyzing the NimBLE controller's codebase, we identified the interrupt service routine (ISR) responsible for updating the connection state machine when a data channel PDU begins [21]. The data structure passed to the ISR contains details about the received packet, including the first bit's arrival time, which is

determined using hardware timers and radio events. For each subsequent connection event, the controller computes the next expected anchor time by adding the connection interval to the actual anchor time captured earlier. Within the ISR, both the actual and expected anchor times for the current event are available. We compute the difference between these two values and update the appropriate bucket of the timing distribution (relative to the anchor time) using the Mynewt RTOS's Stats module. Once the required number of traces for the experiment is collected, the data stored in the stats module is retrieved through a serial terminal for analysis.

5.1 Different Instances of Same Hardware

We first verified if we got similar distributions of packet arrival timings from different instances of the same hardware. We used two nRF52840-DK and two ESP32-WROVER-IE. Plots 1-4 from Fig. 4 present the empirical packet arrival timing distribution from these measurements. We set up the connections with $n = 24$, which gave us a widening window of size $111\mu s$ (computed by the NimBLE stack). Each experiment comprised 50000 measurements. We can see that the traffic forms a distinct cluster without any outliers. We note that we discarded a few outliers (less than 10), as they all had an incorrect CRC. The bounds of the clusters remain consistent across different instances of the same hardware type, and the clusters width stay approximately $5\mu s$, regardless of the hardware used.

5.2 Different Hardware

We then investigated how much the cluster and the widening window size change when we change the used central devices. As explained in section 2.1, the widening window size depends mainly on the parameter n, as well as the SCA of the devices. So, fixing n and taking different devices will show the impact of the crystal inaccuracies on the clusters and the widening window. The nRF52840 has a crystal of 60ppm; the ESP32-WROVER-IE has a 250ppm crystal; and the iPhone 13 also has a 250ppm crystal. A crystal precision is given in ppm, which stands for parts per million. It indicates how much the crystal frequency may deviate from its supposed value. Note that these ppm values were recovered in a real BLE connection using the Apache Mynewt NimBLE BLE stack.

Again, we set up the connection with $n = 24$, which gives a widening window of size $107\mu s$, previously $111\mu s$ on plots 1-4. Results are presented on plots number 5 and 6 from Fig. 4, each experiment consisting of 50000 measurements. The cluster is a bit wider compared to plots 1-4 with a width of $8\mu s$. We can see that taking variations in crystal accuracy have some impact on the cluster as well as the widening window but the cluster remains tight.

Note that the measurement differences between the iPhone13 and the ESP32-WROVER-IE could also come from the BLE stack used. Since the iPhone13's BLE stack is closed source, we have no way to precisely determine if it could be the cause of these variations.

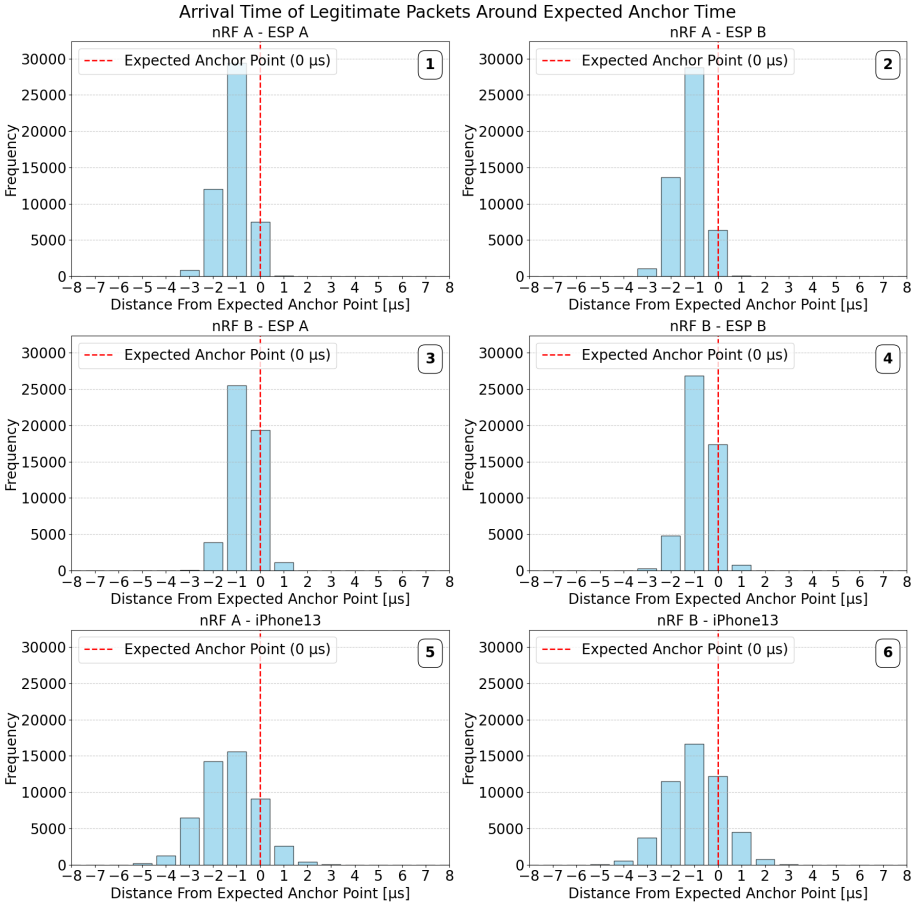

Fig. 4. First-bit arrival time of legitimate packets with different devices. Plots 1-4 show measurements using different instances of the same hardware and plots 5-6 with different hardware. It shows that the arrival time of legitimate packets are tightly clustered and that changing the hardware does not impact much their width.

5.3 Different Connection Parameters

As discussed in Section 2.1, the size of the widening window depends also on the connection interval parameter n. The goal of this experiment is to see how it affects the cluster and the widening window. We used one nRF52840-DK and one ESP32-WROVER-IE, and changed the parameter n to take the following values $\{24, 30, 100, 200, 400, 1000\}$. The number of measurements per experiment was again 5000 and it was reduced to 2500 for $n = 200$, 1200 for $n = 400$ and 1000 for $n = 1000$, as the experiments would have taken too much time to be practically performed since increasing n reduces the rate of packets.

Initially, we measured the arrival time with the two devices being static. To account for more realistic dynamic environments, we repeated the measurements with one device (here the ESP32-WROVER-IE) hand-held and continuously moving around in a 2-4 meter radius around the static device. Note that we were not able to move the device when $n = 1000$, as the connection would timeout quickly. For this parameter value, we present results from measurements with two static devices. We noticed that even though the arrival times of the BLE packets in a dynamic environment are a bit more spread out than in the static one, the differences are insignificant. Note also that there were around 20 other BLE devices detected in the vicinity of the experiment contributing to more challenging, life-like conditions.

The results are presented in Fig. 5, we can see that n is the parameter with the biggest impact on the cluster width as well as the widening window. As n grows, so does the widening window and the width of the cluster (from $5\mu s$ to around $10\mu s$). This is expected since the devices synchronize their clocks when sending packets, so increasing this time makes the clock more prone to drifting. We also notice that as n increases, the cluster slowly shifts away from the expected anchor point but it is insignificant compared to the increase in the size of the widening window. More importantly, starting at $n = 200$, we can see a second cluster appearing (in plot 4 around $-9\mu s$, and plot 5 around $-18\mu s$).

6 Our Detection Method

InjectaBLE exploits the fact that the widening window is frequently too loose in practice, allowing malicious packets to be injected before the legitimate one is received. As discussed in section 2.4, the idea is to shrink the widening window by setting a threshold to filter out the packets arriving too early.

6.1 Threshold Tightness

To propose an effective countermeasure, we first consider the effect of threshold tightness, i.e., what level of protection we get depending on where the threshold is set.

1. **Large space between the threshold and the cluster (i.e., $44\mu s$ or more):** When the distance between the threshold and the first legitimate

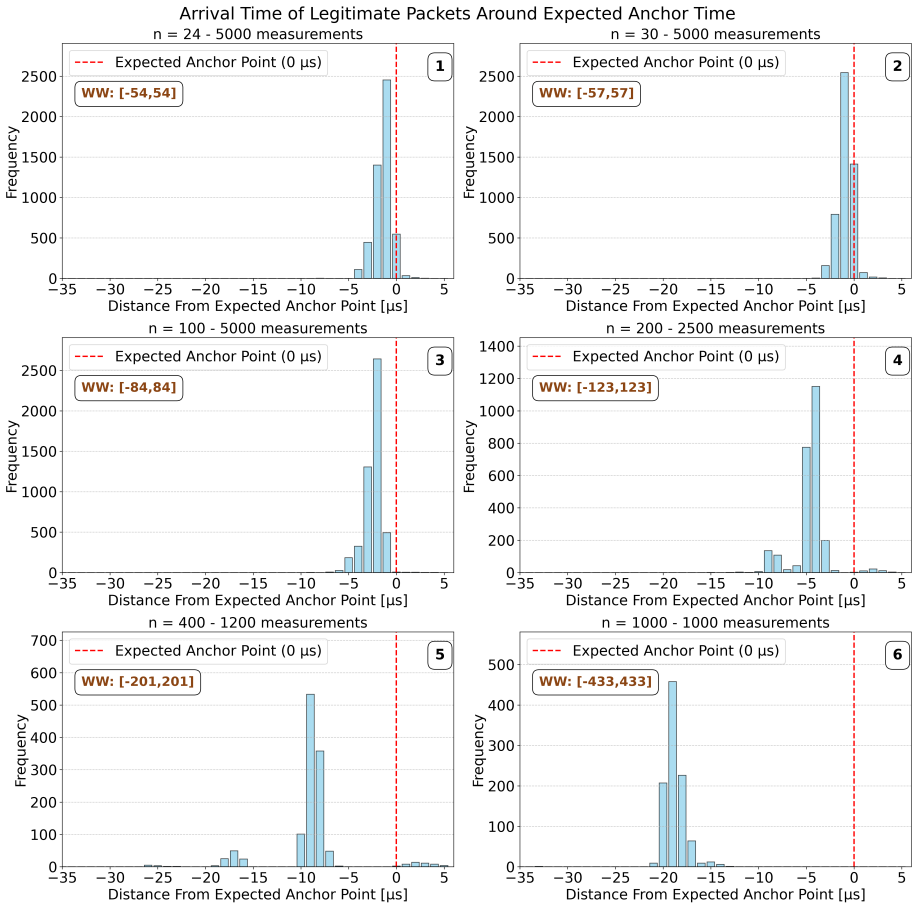

Fig. 5. First-bit arrival time of legitimate packets with different connection interval with one device in motion. The measurements were done with the same hardware but different connection parameters. Specifically, these graphs show the impact of the hop interval. As it increases, we notice that the arrival times are more scattered but always stay in a $35\mu s$ window.

packet arrival time is sufficiently large ($44\mu s$ or more), a precise enough attacker can succeed. This is as in scenario 1 of Fig. 2. In this case, the attacker only needs to position its attack exactly at the threshold to avoid both the mitigation method and a collision with a legitimate packet. The attack will succeed without the need for additional attempts. This scenario is plausible with the OASIS solution, where such a threshold allows attackers to bypass the detection mechanism easily.

2. **Threshold within $44\mu s$ of the first legitimate arrival time:** In this case, the distance between the threshold and the earliest legitimate packet arrival time is less than $44\mu s$. However, legitimate packets may also arrive later, which opens two possibilities. Either the legitimate packet arrival time is sufficiently late to leave open a path for an attacker to inject a malicious packet, meaning the attack is successful and undetected. Alternatively, the legitimate packet arrival time is too early and the attack-injected packet causes a collision, i.e. scenario 2 of Fig. 2. So, the attack success becomes randomized, the exact success probability depending on the arrival time distribution of the legitimate packets. Furthermore, it is important to note that, depending on the handling of flagged packets, the attacker may be able to retry multiple times, increasing the overall attack success probability.
3. **Threshold within $44\mu s$ of the endmost legitimate arrival time:** If the threshold is positioned such that the time between the threshold and the last legitimate packet arrival is less than $44\mu s$, a collision is guaranteed. This scenario provides the highest level of protection, as it ensures that collisions will occur and thus make the likelihood of an attack being successful negligible.

In a nutshell, our method configures the threshold to ensure the third scenario without flagging the early-arriving legitimate packets as false positives. An attacker thus cannot inject a packet without colliding with a legitimate one, but genuine traffic stays undisturbed.

This is possible due to the low spread of the clusters reported in Section 5, with observed cluster spread being no greater than $35\mu s$. Still, it is possible for devices with less precise crystals to be compliant with the BLE specifications but generate a much wider distribution of packet arrival times with a very high connection interval. So, we also need to consider clusters with a larger spread. A threshold that does not disrupt genuine traffic in this case would inevitably fall in the scenario 2 described above, where an attack could still be successful but it becomes probabilistic. These considerations are reflected in our proposed method.

6.2 Implementation Details

CAMEL Threshold In order to set up a relevant threshold, we need to know if this connection falls in scenario 2 or 3 as defined above: whether a $44\mu s$-wide cluster can encompass the data. We do this by collecting the first 255 packets sent after the connection has been established. We then determine the cluster

width by iteratively testing with increasing values, starting with a width of $44\mu s$, until the observed packet arrival times fit within (see below for details). This approach ensures that we find the smallest possible cluster satisfying the given constraints while doing a best effort to ensure the scenario 3 defined above. Then, the threshold is simply defined as the leftmost (earliest) limit of the cluster.

To ascertain that 255 packets are sufficiently representative of the true, underlying timing distribution, we used the Kolmogorov-Smirnov Test (KS-Test). It is a non-parametric test that checks whether two empirical distributions are similar. We used the function *ks_2samp* from SciPy to get the results presented in Table 1. We compared a distribution consisting of 255 new measurements to the ones we presented in Fig. 5. The null hypothesis H_0 is that the two compared distributions are similar, i.e. that 255 packets are enough. With the tested hop intervals, $n \in \{24, 100, 1000\}$, we can see that the null hypothesis is not rejected. We get $D_{KS} < D_{crit}$ with $p > 0.5$.

The possible errors introduced by setting such a threshold are discussed in Section 7.2.

Table 1. Results of KS-test on sub-sampled measurements. **n** is the hop interval. $\mathbf{D_{KS}}$ is the resulting statistic. $\mathbf{D_{crit}}$ is the critical value. We can see that H_0 is not rejected.

n	D_{KS}	D_{crit}	p-value
24	0.03	0.09	0.95
100	0.04	0.09	0.76
1000	0.05	0.09	0.62

Data Collection To represent the packet arrival distribution, a structure called `bucket_buffer` is allocated over 200 bytes in memory. Each bucket, of width $\lceil$`widening_window_size`$/200\rceil\mu s$, occupies 1 byte of memory. For example, if the window size is $150\mu s$, each bucket represents a range of $1\mu s$. This design balances memory efficiency, required on embedded systems, with the precision required for lower window sizes.

For each newly received packet, we calculate the shift from the expected anchor time and update the corresponding bucket. After receiving 255 packets, we execute our threshold computation algorithm.

The time required to collect 255 packets depends mainly on the connection interval: from section 2.1, we know that $connInterval = n \cdot 1.25ms$, so the time required to get 255 packets is $t_{n,255} = n \cdot 1.25 \cdot 255ms$. Since $n \in [6, 3200]$, this data collection step takes at best $t_{6,255} = 1912.5ms < 2s$ and at worst $t_{3200,255} = 1020s$.

Finding The Optimal Cluster We iterate through all possible buckets, starting from the tightest one, and for each bucket, we compute how many outliers

remain if a $44\mu s$ wide cluster is centered at that bucket ($22\mu s$ to the left and $22\mu s$ to the right). The first bucket that results in the minimum number of outliers becomes the center of the cluster, so the complexity in an average case is really low.

If multiple buckets yield the same number of outliers, the earliest-starting one is selected. This strategy accounts for the fact that our data represents a snapshot of genuine traffic, and earlier packets might arrive. By prioritizing leftmost buckets, we maintain a margin for earlier arrivals to avoid false positives while keeping the cluster width at $44\mu s$.

We assume that a $44\mu s$ wide cluster will generally cover the majority of valid packets. However, if no such cluster meets the maximum allowable number of outliers, arbitrarily set to 3 from our experiments, the algorithm incrementally increases the cluster width until the desired minimum number of outliers is reached. This number of outliers is a parameter allowing the adopters to pick a tradeoff between the false positive rate and the false negative rate.

Threshold Calculation Once the optimal cluster is determined, the threshold is simply defined as the start of the cluster, i.e. $center - \frac{width}{2}$, with *center* the center of the cluster and *width* its width.

6.3 Illustration

To illustrate the method, Fig. 6 shows a concrete example of the algorithm's threshold computation using an nRF52840-DK and an ESP32-WROVER-IE connected with a connection interval of 144. These conditions mirror the experimental setup used to show the limitations of the solution proposed by OASIS.

The data, showed in dark blue, creates a tight cluster. So, as expected, we were able to identify a cluster that satisfies the third scenario outlined in Section 6.1, with no compromise on the false positive rate.

6.4 Re-calibration

In order for the threshold to constitute an effective countermeasure, the packets received during the data collection phase, i.e. the first 255 which can take up to $1020s$, must be free of attacker's influence. While in many scenarios, this assumption is reasonable, it does make our method vulnerable in this phase: an attacker could interfere with the calibration by injecting packets during data collection and thereby forcing the system to learn a loose threshold. As a mitigation, we propose an extension, reCAMEL, which recalibrates the threshold regularly so that any attacker-induced distortion is eventually corrected. Performing this on a fixed schedule is possible, but it makes the recalibration times predictable and thus easier for an attacker to exploit. Instead, we adopt a stochastic approach in which recalibration is triggered after a random interval. Each recalibration simply re-collects 255 packets, computes a new threshold, and updates the current one only if the new value is tighter, i.e. it starts closer to the anchor time than

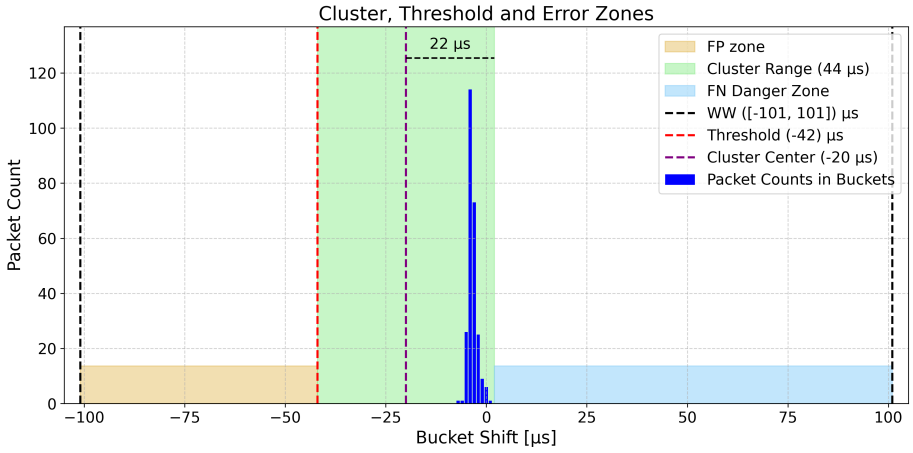

Fig. 6. Illustration of CAMEL's threshold selection. The measurements shown were recovered with a connection interval of 144 ($n = 144$) so as to have a similar setup as in Fig. 3 where we showed OASIS' limitations. The threshold is set at $-42\mu s$, which is $44\mu s$ before the latest packet arrival time with a small margin. This gives no space for an attacker to inject a malicious packet. The two possible error zones: false-positive (FP) and false-negative (FN) zones, discussed in Section 7.2, are also sketched.

the current one. Once this recalibration completes, the system draws a random number from a uniform distribution $\mathcal{U}\{1, 2t-1\}$ to determine how many packets must arrive before the next recalibration starts. This results in an expected interval of t packets between each recalibration. This approach makes the start of the recalibration procedure uncertain enough for the attacker, and allows to control the average overhead introduced.

The experimentally determined computational overhead for a single threshold computation is $1.7ms$ on an nRF52840 running at 64MHz. Neglecting the computation that the device might be doing otherwise, the active time needed for the transmission of a very short PDU on the same platform is typically around $0.5ms$. The conservative average overhead introduced by recalibration can therefore be expressed as $1.7ms/(t \cdot 0.5ms)$. When we fix the desired overhead at $\leq 0.1\%$, we get $t = 3400$. This means that the longest wait between two recalibrations will correspond to 6799 packets, or 204 seconds for a connection interval of $30ms$ ($n = 24$), which is relatively frequent.

This design forces an attacker aiming to maintain a loose threshold to remain in proximity and continuously interfere with each recalibration attempt, especially because it cannot content itself with tampering with short bursts of traffic because recalibration schedule is unpredictable. Fig. 7 shows the pseudo-code for this recalibration process.

This approach pragmatically hardens the crucial part of setting the right threshold. Since the new threshold can only increase, it cannot be exploited to loosen the margin for packet injection. Likewise, an attacker cannot tighten

```
1: function RecalibrationProcedure(t)
2:     collect 255 datapoints
3:     compute new_threshold
4:     if new_threshold > threshold then
5:         threshold = new_threshold
6:     end if
7:     packets_to_wait ←$ U{1, 2t − 1}
8:     return packets_to_wait
9: end function
```

Fig. 7. Re-calibration algorithm.

the threshold to induce false positives: the attacker would have to send packets arriving later in the widening window and hence would create collisions which would restart the connection.

Although the initial calibration remains susceptible to manipulation, the proposed recalibration mechanism introduces significant practical hurdles for persistent attackers. They must repeatedly compromise each recalibration attempt; otherwise, the threshold will revert to a secure value.

Overall, the overhead incurred by the recalibration is negligible. The memory footprint is dominated by a 256B array storing the timings used to recalibrate, while the expected computational overhead is 0.1% as per our example.

7 Discussion

7.1 Packet Collision

As shown in Fig. 2, and discussed in section 6.1, we leverage packet collision as a tool to mitigate the InjectaBLE attack without disrupting the genuine traffic, in a model where a packet collision is assumed to make both colliding packets undecodable. This model is based on the observation that, since the packets arrive at the same time and on the same frequency, the packet arriving second, the one colliding, will perturb the signal.

This perturbation depends on one factor: the ratio of signal strength between the first packet and the colliding packet. If the colliding packet arrives at the receiver with considerably less power than the first packet, then it will not disturb the communication and will only be perceived as noise. But if on the other hand it arrives with sufficient power, both packets will become undecodable by the receiver. This latter possibility can be seen as a DoS. The power of a received signal is dependent on multiple factors: the signal strength at the transmitter, the physical proximity between the devices, and whether or not there are physical obstacles in the way (wall, post, etc.).

Our method of mitigating packet injection will either completely stop the attack, or will downgrade it to a DoS. Furthermore, this solution reduces the accessibility of such an attack by imposing more constraints on the attacker. Indeed, it forces him to use more sophisticated equipment due to the tight timing

requirements, as well as find solutions to the physical constraints like coming closer to the target. Even then, he needs the legitimate devices to use specific communication parameters with bad enough clock accuracies which would put our method in the scenario 2 from section 6.1, and the attack success would be only probabilistic depending on where the genuine packet would arrive in the widening window.

Our collision model could be invalidated when a receiver exhibits a *capture effect* and decodes the stronger of two overlapping BLE packets. Controlled studies on BLE's 1 Mb/s PHY and related concurrent-transmission settings report capture success predominantly as a function of signal-power asymmetry: success rates are near-zero below ≈ 2 dB, around $\approx 50\%$ at ≈ 4 dB, and exceed 90% beyond ≈ 7 dB under near-simultaneous overlaps within the preamble window [22, 23]. Although reproducing such timing and power advantages reliably with unmodified commodity hardware is non-trivial, any adversary who does sustain the required power difference will unavoidably induce a statistical signature: a persistent increase in RSSI on first-arrivals and per-channel skew relative to the baseline, which can be detected with lightweight monitors [24]. Conversely, if the adversary keeps the power difference small to avoid RSSI elevation, the attack necessarily devolves into destructive collisions, yielding bursts of missed frames and packet-drop-rate anomalies. As packet capture is a physical-layer phenomenon independent of our work, we treat these RSSI/packet-drop-rate monitors as complementary to CAMEL and, for tractability of analysis, abstract away capture while assuming commodity-scale adversaries are detectably bounded by either RSSI-based or collision-based signals.

In conclusion, the packet-injection problem has two axes: one axis is addressed by our solution, CAMEL, while the other, physical-layer attacks exploiting stronger RSSI, lies outside our threat model and remains an open problem. Ultimately, combining both axes would converge toward a lightweight IoT IDS incorporating CAMEL and RSSI as an additional input signal.

7.2 Detection Error Prediction

We analyze the likelihood of false-positives (a legitimate packet erroneously flagged as anomalous) and false-negatives (a malicious packet not detected). None of these events have been observed in our experiments presented in Section 5. We therefore establish upper bounds for these errors beyond in the subsequent analysis.

A *false positive* occurs if a legitimate packet arrives much earlier than expected, i.e. in the yellow zone on Fig. 6, which leads to the packet being lost. Due to the absence of a false positive in the empirical data from our experiments, we compute a conservative upper-bound using the rule of three [25]: if an event did not occur in k trials, we can upper-bound the probability of the event with 95% confidence as $p \leq 3/k$. We collected $k = 5000$ measurements for the connection interval $n \in \{24, 30, 100\}$, yielding False-Positive Rate $FPR \leq 0.06\%$ (6).

$$FPR \leq \frac{3}{k} = 0.06\% \tag{6}$$

For the connection interval $n = 1000$, we only collected $k = 1000$ measurements, suggesting a bound of $FPR \leq 0.3\%$. However, here, the number of measurements is reduced because the experiment execution is slowed down by an order of magnitude due to the large connection interval.

We turn to *false negatives.* In our model, a malicious packet that arrives in the green zone shown in Fig. 6 would result in a failed attack, since it would either be stopped by the threshold or collide with a genuine packet. Then, the only possibility for a malicious attack to be received without detection is when a legitimate packet arrives in the FN danger zone (blue zone). With this, we can upper bound the False Negative Rate (FNR) as the ratio of legitimate packets arriving in the FN zone l versus the total number of packets collected k (7).

$$FNR = \frac{l}{k} \tag{7}$$

Having observed no legitimate frame in the danger zone in the collected data, we again apply the rule-of-three to get a conservative upper-bound on the FNR.

For the connection interval $n \in \{24, 30, 100\}$, with $k = 5000$, this yields $FNR \leq 0.06\%$ (8) and $FNR \leq 0.3\%$ for $n = 1000$ with $k = 1000$.

$$FNR \leq \frac{3}{k} = 0.06\% \tag{8}$$

As presented in Fig. 5, we could not try with $n > 1000$ since the BLE devices we used were not able to hold a connection with such a big connection interval.

While our evaluation demonstrates low false-positive and false-negative rates, we acknowledge that our data collection was not performed across varied RF environments. The measurements were not acquired in a controlled test facility such as an anechoic chamber, but rather in an ordinary room with substantial RF activity, multipath reflections, and interference typical of indoor BLE deployments. Although this setting represents a realistic everyday environment, it does not capture the full diversity of RF conditions that may occur in the wild.

A comprehensive exploration of packet's arrival time under highly variable or adversarial RF conditions remains an open problem. This challenge is non-trivial and largely unaddressed in prior BLE security and timing-based studies, which similarly evaluate their systems under a limited set of RF conditions. While expanding this analysis would be valuable, it constitutes a sizable research effort in its own right; effectively a standalone topic beyond the scope of this work.

8 Conclusion and Future Work

Spoofing attacks on unencrypted Bluetooth Low Energy (BLE) communication channels represent a real-world security threat. The number of BLE devices using no encryption remaining stubbornly huge even to date reminds us that the decision to (not) use link encryption is in the application developer's purview, indicating that this issue cannot be waved-off as a temporary problem and requires a complementary solution.

Our offensive contribution highlights that the injection detection component of the landmark BLE IDS OASIS can be bypassed in many connection configurations (the selection of which is also in the purview of the application developer), stressing the need for a more adaptive solution. We stress that intrusion resistance is conjunctive: a single undetected path to MitM collapses the overall protection claim, regardless of how many other attack classes are covered. Covering this gap, our defensive result is an adaptive, automated threshold determination method for injection detection dubbed CAMEL, ensuring that security from moderately powerful attackers across all configurations and across diverse network environments.

While powerful adversaries capable of transmitting at high power may be able to shout over legitimate traffic, rendering CAMEL ineffective, this either forces the attacker to use more expensive equipment (making the attack economically non-viable in many applications) or move physically closer to the target (reducing attack practicality and increasing exposure to detection).

Investigating signal collisions in Bluetooth Low Energy (BLE) environments is a promising direction for understanding and expanding our threat model, and the design of portable and resource-efficient detection covering a broader range of attacks. Utilizing specialized hardware to examine the signal power thresholds where normal traffic becomes disrupted or overtaken could provide valuable insights into the interactions between legitimate and malicious signals. Exploring these thresholds across diverse configurations may reveal patterns that contribute to developing more effective strategies for mitigating signal-based attacks, such as BLE packet injection.

Acknowledgment

This research was co-funded by the European Union's Horizon Europe research and innovation program under grant agreement No. 101135982 (HYPER-AI).

References

1. Bluetooth SIG, "2024 Bluetooth Market Update | Bluetooth Technology Website," https://www.bluetooth.com/2024-market-update/, [Accessed 09-01-2025].
2. R. Cayre, F. Galtier, G. Auriol, V. Nicomette, M. Kaâniche, and G. Marconato, "InjectaBLE: Injecting malicious traffic into established bluetooth low energy connections," in *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2021, pp. 388–399.
3. 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)*, Berlin, Germany, Oct. 2019.
4. "Xiaomi mi band 2 security." [Online]. Available: https://www.iot-tests.org/2018/08/in-the-security-check-xiaomi-mi-band-2/
5. "Reverse engineering a bluetooth lightbulb." [Online]. Available: https://blog.wokwi.com/reverse-engineering-a-bluetooth-lightbulb/

6. "Theengs openmqttgateway." [Online]. Available: https://docs.openmqttgateway.com/prerequisites/devices.html#for-ble-devices
7. H. C. Pöhls and L. Steffens, "Checking the impact of security standardization — a case study on bluetooth le pairing of internet-of-things devices," in *ICT Systems Security and Privacy Protection*, L. Nemec Zlatolas, K. Rannenberg, T. Welzer, and J. Garcia-Alfaro, Eds. Cham: Springer Nature Switzerland, 2025, pp. 49–63.
8. R. Cayre, V. Nicomette, G. Auriol, M. Kaâniche, and A. Francillon, "OASIS: An intrusion detection system embedded in bluetooth low energy controllers," in *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security*, ser. ASIA CCS '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 700–715. [Online]. Available: https://doi.org/10.1145/3634737.3645004
9. A. Famera, B. Hilger, S. Bhunia, and P. Heil, "Analyzing the mirai iot botnet and its recent variants: Satori, mukashi, moobot, and sonic," 2025. [Online]. Available: https://arxiv.org/abs/2508.01909
10. K. Boeckl *et al.*, "Considerations for managing internet of things (iot) cybersecurity and privacy risks," NIST IR 8228, Tech. Rep., 2019.
11. 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.
12. J. Wu, Y. Nan, V. Kumar, D. J. Tian, A. Bianchi, M. Payer, and D. Xu, "BLESA: Spoofing attacks against reconnections in bluetooth low energy," in *14th USENIX Workshop on Offensive Technologies (WOOT 20)*. USENIX Association, Aug. 2020.
13. 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)*. USENIX Association, Oct. 2020, pp. 397–411.
14. M. Yaseen, W. Iqbal, I. Rashid, H. Abbas, M. Mohsin, K. Saleem, and Y. Bangash, "MARC: A novel framework for detecting mitm attacks in ehealthcare ble systems," *Journal of Medical Systems*, vol. 43, 10 2019.
15. 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)*, 2018, pp. 1–8.
16. F. Galtier, R. Cayre, G. Auriol, M. Kâaniche, and V. Nicomette, "A psd-based fingerprinting approach to detect iot device spoofing," in *2020 IEEE 25th Pacific Rim International Symposium on Dependable Computing (PRDC)*, 2020, pp. 40–49.
17. G. Benita, L. Sestrem, M. E. Garbelini, S. Chattopadhyay, S. Sun, and E. Kurniawan, "Vaktble: A benevolent man-in-the-middle bridge to guard against malevolent ble connections," in *2024 Annual Computer Security Applications Conference (ACSAC)*, 2024, pp. 621–635.
18. Bluetooth core specification v6.0. [Online]. Available: https://www.bluetooth.com/specifications/specs/core60-html/
19. R. Cayre, V. Nicomette, G. Auriol, M. Kaâniche, and A. Francillon, "OASIS GitHub repository," 2024. [Online]. Available: https://github.com/RCayre/oasis/blob/main/modules/injectable/module.c

20. R. Cayre, F. Galtier, G. Auriol, V. Nicomette, M. Kaâniche, and G. Marconato, "InjectaBLE Firmware GitHub repository." [Online]. Available: https://github.com/RCayre/injectable-firmware
21. GitHub repository Apache Mynewt NimBLE pointing to the function responsible for updating the connection state machine. [Online]. Available: https://github.com/apache/mynewt-nimble/blob/master/nimble/controller/src/ble_ll_conn.c
22. C. Roest, "Enabling the chaos networking primitive on bluetooth le," Master's thesis, TU Delft, 2015. [Online]. Available: https://resolver.tudelft.nl/uuid:95f50b13-6af5-4bb3-83ef-84b065e13682
23. B. A. Nahas, A. Escobar-Molero, J. Klaue, S. Duquennoy, and O. Landsiedel, "Blueflood: Concurrent transmissions for multi-hop bluetooth 5—modeling and evaluation," *ACM Trans. Internet Things*, vol. 2, no. 4, Jul. 2021. [Online]. Available: https://doi.org/10.1145/3462755
24. S. Roth and A. Sezgin, "Anomaly detection for sensing security," 2025. [Online]. Available: https://arxiv.org/abs/2506.10718
25. E. Eypasch, R. Lefering, C. Kum, and H. Troidl, "Probability of adverse events that have not yet occurred: A statistical reminder," *BMJ (Clinical research ed.)*, vol. 311, pp. 619–20, 09 1995.

A Appendix

A.1 OASIS Code Interpretation

As mentioned in section 4.1, since the threshold configuration was not detailed in the OASIS paper [8], we had to interpret the available code [19].

First, the connection interval is recovered (defined in this paper as $t_{act,i+1}$), and the absolute value of the difference with the median is computed and called *absolute_ error*. Then, the following line scales this error:

$$absolute\_\,error = (2000 * absolute\_\,error)/median \tag{9}$$

Finally, the condition to raise an alert is defined as:

$$absolute\_\,error \leq threshold \text{ AND } absolute\_\,error > 0 \tag{10}$$

with $threshold = 10$.

We noticed that the condition on the absolute error to raise an alert is counter-intuitive and should be $absolute\_\,error > threshold$. So, we found 2 possible interpretations:

1. **The code is correct as is:** In this case, the condition $absolute\_\,error \leq threshold$ serves no purpose and only the second one, $absolute\_\,error > 0$, is relevant. This second condition is interesting due to the fact that $absolute\_\,error$ is defined as an unsigned integer. Since the last update of the variable $absolute\_\,error$ is Eq. 9, we get that the condition $absolute\_\,error > 0$ is equivalent to
$$\frac{2000 * absolute\_\,error}{median} > 1 \tag{11}$$
which is equal to the one we used as Eq. 1. From this assumption we derived the computations in section 4.3 and found the condition $n \geq 71$.

2. **The condition should be changed:** Here we assume that there is a mistake in the code and that the condition to raise an alert should be $absolute\_error > threshold$, which can be re-written using Eq. 9 as

$$\frac{2000 * absolute\_error}{median} > 10 \tag{12}$$

Note that in this case the second part of the condition to raise an alert, i.e. $absolute\_error > 0$, serves no purpose anymore. Doing the same computations as in section 4.3, we get the condition $n \geq 8$.

Even though the latter possibility seems to be the most logical to follow, it is not the solution used as a countermeasure in OASIS where they presented strong results. Furthermore, the first possibility gives us the most restrictive condition, i.e. $n \geq 71$, so we decided to report this one in our paper.