# SIROCCO: A dry wind to warn you from Bluetooth attacks

Paul L. R. Olivier*, Florent Galtier*, Guillaume Auriol*†, Vincent Nicomette*†, Romain Cayre*†
*CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France
Email: name.surname@laas.fr
†Univ de Toulouse, INSA, LAAS, F-31400 Toulouse, France
Email: name.surname@insa-toulouse.fr

***Abstract*—Bluetooth Low Energy (BLE) has become ubiquitous in Internet of Things (IoT) devices, yet it remains vulnerable to sophisticated over-the-air attacks that can compromise data security. This paper introduces Sirocco, a novel host-based intrusion detection system that leverages the open source Zephyr RTOS to address critical security challenges in BLE communications. Sirocco provides a flexible and lightweight detection framework that integrates with the BLE protocol stack. It targets sophisticated and low-level attacks such as spoofing, signal injection, and key sharing vulnerabilities. By inserting lightweight metric collection functions directly into the BLE stack's interrupt service routines and employing a dedicated kernel thread for analysis, Sirocco minimizes performance overhead while maintaining real-time attack detection capabilities across different device roles. Experimental evaluation demonstrates the system's effectiveness, with the framework introducing minimal performance and power consumption overhead.**

## I. Introduction

Although Internet of Things was forecast in the 80s, it is only in the last fifteen years or so that we have been able to acknowledge their omnipresence in our everyday life. Despite this, a number of security issues need to be addressed in their deployment. Connectivity is one of them. Looking at the large number of wireless protocols that have been specified, deployed, and abandoned, it becomes evident that finding the right balance between performance, energy efficiency, security, and market viability remains a significant challenge.

Among these protocols, Bluetooth Low Energy (BLE) has emerged as a dominant standard due to its energy efficiency and adoption in a wide range of devices and applications. However, despite its advantages, it is not immune to security vulnerabilities. Numerous flaws, both in the specifications and in its implementations, have allowed spoofing, breaking cryptography, privilege escalation, and remote code execution [29], [9], [25], [6], [28], [13], [5], [18], [8], [7].

The specification of the BLE protocol suffers from inherent weaknesses that are difficult to prevent, as these vulnerabilities are embedded within the design itself [29], [6], [9]. Moreover, the specification includes a large set of possible configurations, which make the implementation cumbersome and more error-prone. A typical example is the check of I/O capabilities during the pairing process. Depending on whether the device has a display, a keyboard or no means of input or output, the pairing method used will be more or less secure. While the goal is to cover a large set of use cases, it results in increasing the complexity for the configuration and implementation. This is confirmed by the newer specifications (from version 4.2) calling it *legacy pairing* while new pairing method are called *secure pairing*. In addition, legacy pairing are still used in today products despite the common knowledge of their insecurity [20], [29]. In practice, this extended set of choices makes the implementation easier to be wrongly configured by the vendor, resulting in selling insecure devices.

We observed that many BLE attacks exploit vulnerabilities in multiple layers of the BLE protocol stack [29]. This is explained by its architectural design divided between the host and the radio controller. In particular, the radio controller operates under strict constraints: it must function within the 2.4 GHz ISM band, an environment prone to interference and noise, while maintaining low power consumption. These radio restrictions complicate the implementation of robust physical layer security mechanisms. Additionally, the resource limitations of BLE devices can lead to compromises in the implementation of security features, such as the use of shorter cryptographic keys to reduce computational overhead.

Typically, attackers start by targeting the physical or link layers. Techniques such as radio spoofing and signal injection allow the establishment of unauthorized connections or the setting of a Man-in-the-Middle (MITM) between legitimate devices [18], [8], [7], [9]. Once this foothold is gained, attackers move to compromise the security mechanisms in the higher layers of the stack. They may bypass [3] or downgrade encryption [6], [4] and authentication protocols, exploiting implementation flaws or misconfigurations [30], [2], [14] that are common in BLE devices. This enables them to gain unauthorized access to the application layer, where they can read, modify, or exfiltrate sensitive data [25], [28], [5].

The layered nature of BLE, combined with the variability in security implementations across devices, underscores the need for robust defenses. Formal verification techniques have been applied to analyze Bluetooth specifications and implementations [28], [23], [30]. Program analysis, and fuzzing in particular, have proven effective in discovering bugs in the various BLE implementations [17], [16], [21], [26], [19]. However, these preventive measures have limitations in addressing the security threats once the device is deployed.

Runtime Intrusion Detection Systems (IDS) are crucial for identifying and mitigating sophisticated attack chains, as they offer a proactive approach to securing BLE communications. IDS are often categorized based on where they collect their metrics: either on the *network* or on the *host*. However, BLE presents unique challenges that make traditional network IDS approaches more difficult to implement effectively. For instance, BLE uses adaptive frequency hopping to lower the probability of collisions in the crowded 2.4 GHz band.

This makes it challenging for external probes to consistently monitor a given connection. Additionally, BLE networks are inherently mobile where devices may change their physical locations. This mobility further complicates the deployment of static network probes, as they have a limited range.

Given these challenges, host-based IDS approaches provide a more practical solution to monitor BLE traffic. By embedding intrusion detection mechanisms directly within BLE devices, many of the drawbacks associated with external probes can be mitigated. It ensures continuous monitoring regardless of a device's location or the specific frequency channel in use. However, implementing such an IDS for BLE comes with its own set of challenges. Many attacks target the lower layers of the BLE stack, and detecting such attacks may require firmware modifications, a complex task that risks affecting device stability. Moreover, the resource constraints pose additional challenges in implementing comprehensive IDS solutions without affecting performance or battery life.

With these challenges in mind, we propose `Sirocco`, a novel host-based IDS built on Zephyr's open-source Bluetooth Low Energy (BLE) stack. Our approach offers several key advantages. First, it enables seamless integration with existing device applications by leveraging a modular architecture that adapts to both central and peripheral BLE device roles. Second, it delivers real-time attack detection and alerts with customizable response mechanisms. Finally, it minimizes performance overhead while maintaining system stability through careful separation of metric collection and analysis processes. Experimental evaluation demonstrates `Sirocco`'s effectiveness with low latency, minimal power consumption, and limited memory footprint across diverse BLE device roles. We release `Sirocco`'s source code at https://github.com/plrolivier/sirocco.

## II. Background

Bluetooth Low Energy (BLE) is a wireless protocol designed for low-power and short-range communication. Introduced in Bluetooth 4.0, it offers a cost-effective and energy-efficient solution for Internet of Things (IoT) devices.

### A. Roles

BLE defines several roles that devices can assume.

- **Central:** A central device initiates connections with peripheral devices. Typically, it has more processing power and memory, allowing it to manage multiple connections at the same time. For example, a smartphone often acts as a central device when interacting with BLE peripherals like fitness trackers or smart home devices.
- **Peripheral:** A peripheral device advertises its presence and awaits connection requests from central devices. These devices are usually resource-constrained, with lower processing power and memory. Examples include sensors, beacons, and wearable devices.

Other roles include the **broadcaster**, which sends out non-connectable advertisements, and the **observer**, which listens to these advertisements without initiating a connection.

*Advertisements* are fundamental to BLE's communication model. They are short broadcasts sent periodically by a peripheral device to announce its presence and capabilities. BLE advertisements can carry small amounts of data, allowing them to transmit information such as the device's name, supported services, or connection parameters. Devices in observer or central roles scan these advertisements to discover peripherals for potential connections.

### B. BLE Security Features

Bluetooth Low Energy (BLE) faces several threats, including *identity tracking*, *passive*, and *active eavesdropping*. Identity tracking exploits static Bluetooth addresses to monitor device activity. Passive eavesdropping allows attackers to intercept unencrypted data. This highlights the need for secure key generation and exchange. Active eavesdropping or Man-In-The-Middle (MITM) attacks involve an attacker impersonating legitimate devices to intercept communications. Mitigating this requires robust authentication to ensure devices connect to verified counterparts.

To protect the protocols against them, BLE defines its *pairing process* to generate, distribute and authenticate its cryptographic key to encrypt the communication. The pairing process is composed of three phases:

- **Phase 1: Pairing Initiation.** During this phase, the devices agree on the pairing methods that will be used in the subsequent phase and the security level of the connection. For that, they first use the OOB and MITM flags before their I/O (Input/Output) capabilities. The different combinations can be found in [24].
- **Phase 2: Pairing and Key Generation.** Based on the previous choice, devices select an appropriate association model. BLE supports several association models, including *Just Works*, *Passkey Entry*, *Numeric Comparison*, and *Out-of-Band*. The choice of association model directly influences the security of the generated keys.
- **Phase 3: Key Distribution.** In this final phase, the LTK, along with other cryptographic keys (e.g., Identity Resolving Key (IRK) and Connection Signature Resolving Key (CSRK)), is distributed between devices. These keys are crucial for securely reconnecting and re-encrypting subsequent communications.

During pairing, the devices may choose between two methods: *Legacy* pairing or *LE Secure Connections* pairing. In Legacy Pairing, a Temporary Key (TK) is exchanged and used to generate a Short-Term Key (STK), which encrypts the initial connection. However, the STK is vulnerable to brute-force attacks due to its limited lifespan and exposure during the pairing process [22]. It is no more recommended using, but many devices keep supporting it for compatibility reasons.

LE Secure Connections, introduced in Bluetooth 4.2, significantly improve security by using Elliptic-Curve Diffie-Hellman (ECDH) cryptography. This method generates a Long-Term Key (LTK) instead of an STK, offering enhanced protection. The LTK is used for encrypting future connections between the devices, ensuring that subsequent sessions can be re-established securely without repeating the pairing process.

The use of the LTK enables robust protection against eavesdropping and replay attacks, as it is exchanged securely and stored for ongoing use. However, its implementation offers new attack vectors [28], [3].

*Authentication* strength is directly influenced by the association model used. Just Works (JW) involves minimal user interaction and does not provide authentication, making it vulnerable to MITM attacks. It is primarily used when devices lack input/output capabilities. Passkey Entry (PE) displays a 6-digit passkey on one device, to be entered on the other. While this method provides basic authentication, it can be vulnerable to passive eavesdropping if not combined with Secure Connections. In Out-of-Band (OOB), keys are exchanged through an external medium like NFC, providing strong security as long as the OOB channel is secure. Numeric Comparison (NC) requires users to confirm matching numbers on both devices. It is exclusive to LE Secure Connections.

*Services* are organized sets of characteristics that define a specific function or feature of a device. Each service is identified by a unique 16-bit or 128-bit UUID (Universally Unique Identifier). A service may include multiple characteristics, each representing a piece of data or control point, such as a heart rate measurement or battery level. Services are protected with access controls: *No Security* (NS), *Encryption Required* (ER), *Authentication Required* (AR) and *Encryption and Authentication required* (AE).

### C. BLE Protocol Stack Architecture

The BLE protocol stack is divided into two components: the *Host* and the *Controller*. The Host manages higher-level functions such as device discovery, connection management, and security protocols. The Controller operates at a lower level, handling *real-time* communication tasks, including managing the Link Layer (LL) and Physical Layer. It is responsible for packet transmission, reception, and the execution of control procedures that ensure reliable data delivery. The *Host-Controller Interface* (HCI) is a standardized protocol that facilitates communication between the Host and Controller, enabling flexibility in hardware design. HCI allows for interoperability between components from different vendors and is crucial for maintaining a modular architecture.

In practical implementations, BLE devices are categorized into *single-chip* and *dual-chip* architectures. Single-chip architectures, where both the Host and Controller are integrated into one unit, are common in peripherals prioritizing low power and compact design. On the other hand, dual-chip architectures separate the Host and Controller into distinct hardware components, typically used in central devices like smartphones, which demand higher performance and flexibility.

## III. Motivations

### A. Threat Model

In our threat model, we assume an attacker whose ultimate goal is to gain unauthorized access to data exchanged during BLE communication. For this attacker, we adopt the Dolev-Yao attack model, where the attacker has full control over the communication channel but lacks physical access to the targeted devices. For instance, the attacker can impersonate legitimate devices, eavesdrop on both ongoing and newly established BLE communications, and transmit arbitrary signals to disrupt or hijack connections. However, we do not include any exploitation of vulnerabilities related to the implementation such as memory corruption, overflows, etc.

TABLE I
OTA BLE ATTACKS GROUPED BY TYPES

| Layers | Attack Type | Attack | Affected Phase | | |
|---|---|---|---|---|---|
| | | | DD | KS | DE |
| Controller | Radio spoofing resulting in MITM | *BTLEJuice* [7] | ● | | |
| | | *GATTacker* [18] | ● | | |
| | Signal injection | *BTLEJack* [8] | | | ● |
| | | InjectaBLE [9] | | | ● |
| | Breaking key sharing | BlueMirror [15] | | ● | ◐ |
| | Breaking key encryption | *KNOB* [6], [4] | | ● | ◐ |
| Host | Illegal service access | BLESA [28] | | | ● |
| | | BlueDoor [25] | | | ● |

*italic*: a detection module is implemented in Sirocco.
DD: Device Discovery, KS: Key Sharing, DE: Data Exchange
●: targeted by the attack - ◐: affected by the attack

### B. Attacks in scope

We chose to focus on five different types of Over-the-Air (OTA) attacks. We have adopted the classification proposed by Wu et al. [29]. For each type of attacks, we identified an implementation in particular. Table I summarizes our choices.

*1) Spoofing:* Spoofing involves playing with advertisements to mislead the Central into connecting to the attacker. For instance, BTLEJuice [7] first connects to the peripheral which makes it stop advertising. Then, it is possible for the attacker to start advertising in place of the legitimate Peripheral and wait for the Central to connect. In contrast, GATTacker [18] prefers to send advertisements faster than the legitimate peripheral. This increases the probability for the Central to receive the attacker advertisements and connect to it instead of the legitimate peripheral. All the attacker has left to do is to connect to the legitimate peripheral and forward packets.

*2) Signal Injection:* Signal injection refers to an attack where an adversary deliberately inserts or modifies radio signals to interfere with. In general for BLE, it is used to hijack an established connection to replace one of the legitimate/rightful communicator with the attacker. Because of the frequency hopping mechanism used during a connection, BTLEJack [8] proposes first to synchronize with the established connection in order to jam the signal for the connection to drop. After that, it is possible for the attacker to hijack the Central role if the peripheral does not disconnect or terminate the connection.

InjectaBLE [9] exploits the window widening mechanism so that attacker packets are received before the legitimate packets. It is therefore possible to hijack or MITM a connection. The window widening is the increase in listening time for a received packet. It exists because of clock accuracies in which packets are sent, so the recipient shall adjust its listening time to it (Bluetooth Core Specification, Vol 6, Part B, 4.2.4 [24])

*3) Breaking Key Sharing:* Key sharing attacks target the pairing process in BLE to compromise the exchange of

cryptographic keys between devices. These attacks exploit vulnerabilities in the pairing protocols to either intercept the shared keys or trick devices into pairing with an attacker.

The BlueMirror [15] attacks show how this can be done by exploiting the Passkey Entry association model for BLE. The attacker intercepts and reflects messages between devices during pairing. By doing this, they can trick one device into thinking it's paired with a legitimate partner. In a more advanced version, the attacker can even learn the secret passkey and use it to gain full access to the encrypted communication. These attacks work because the pairing protocol doesn't properly authenticate messages. This weakness allows an attacker to insert themselves into the pairing process, potentially gaining unauthorized access to BLE communications.

*4) Breaking key encryption:* Key encryption attacks focus on weakening or breaking the encryption mechanisms used to protect BLE communications. The KNOB [6], [4] attacks is a typical example of this category. KNOB exploits a vulnerability in the Bluetooth specification that allows an attacker to manipulate the entropy negotiation of the encryption key. By forcing both devices to reduce the encryption key length, the attacker can easily brute-force the key to break the encryption.

*5) Illegal Service Access:* Illegal service access attacks exploits the weaknesses of the specification and the implementations to obtain the data exchanged by the Bluetooth communication. It is the ultimate goal in case of our attack model, and is obtained by carefully chaining attacks on "lower" layers.

Usually, these types of attacks are based on the assumption the attacker managed to connect with the legitimate device via spoofing or signal injection, and after that, try to abuse the protocol by lowering the encryption and authentication through tricks. From that, it is possible then to illegally access the services and exchanged data. A typical example is the Bluedoor [25] which chain multiple attacks (spoofing, MITM, encryption downgrade). BLESA [28] spoofs the returned attribute value instead of returning an error message.

## IV. Detection approaches

### A. BLE Intrusion Detection Systems

Several research works have been published to enhance the security of BLE networks. They can be categorized based on where they collect their metrics: either on the network with external probes or on the host itself by hooking primitives. In this section, we review three significant contributions, that we consider as representative of these two approaches : BlueShield [27] (external probe detection), OASIS [11], and BlueSWAT [12] (embedded detection).

BlueShield [27] presents a device-agnostic, non-intrusive monitoring framework for detecting spoofing attacks in stationary BLE networks. It detects anomalies in the physical features of advertising packets (advertising interval, Carrier Frequency Offset (CFO), Received Signal Strength Indicator (RSSI)). BlueShield operates in two phases: profiling and monitoring. During the offline profiling phase, it records and analyzes advertising packets to extract device characteristics and physical features. In the runtime monitoring phase, BlueShield employs three distinct probes using a randomized channel switching mechanism across the three BLE advertising channels. The framework employs statistical models to detect anomalies in the physical features, comparing observed patterns against profiles established during the profiling phase to identify potential spoofing attacks.

OASIS [11] proposes an Intrusion Detection System (IDS) embedded directly into BLE controllers. It focuses on detecting low-level attacks such as spoofing, signal injection and Man-in-the-Middle (MITM). Unlike external monitoring solutions, OASIS instruments the controller firmware itself, providing a more in-depth view of BLE communications. The system patches the firmware binary to insert hooks at critical points in the BLE controller stack. These hooks allow OASIS to collect a range of metrics such as advertising intervals, RSSI, and packet integrity information. Several detection modules are implemented that analyze the collected metrics to identify potential attacks, including BTLEJuice, GATTacker, BTLEJack, InjectaBLE and KNOB attacks. By operating at the controller level, OASIS can detect attacks in real-time, even in the connected mode of BLE operation, which is typically challenging for external monitoring systems.

BlueSWAT [12] introduces a lightweight state-aware security framework for BLE, focusing on mitigating session-based attacks. It employs a finite state machine (FSM) to monitor sequential actions of BLE connections at runtime. The FSM is abstracted from the Bluetooth Specification, modeling various attack patterns as malicious transitions. To capture session states and update the FSM in real-time, BlueSWAT hooks the Link Layer (LL) and Security Manager Protocol (SMP) of the BLE stacks. BlueSWAT leverages a lightweight eBPF virtual machine to run its detection with minimal performance impact and across different BLE architectures and stacks. When a session triggers a transition, BlueSWAT loads the corresponding eBPF programs from a policy cache for verification.

### B. Comparison

BlueShield's effectiveness is limited by its focus on stationary BLE networks, excluding many mobile BLE devices such as wearables and portable medical equipment. BlueShield targets spoofing attacks, which overlook other types of BLE vulnerabilities. OASIS focuses on low-level attacks related to protocol design such as signal injection, spoofind and MITM attacks. BLueSWAT addresses both design flaws and higher-levels attacks, including function errors (authentication bypass, key compromise, encryption failure) and runtime errors (buffer overflow, logic error). Its state-aware approach allows for detection of complex, multi-step attack sequences, which are harder to identify using other approaches.

As an external monitoring solution, BlueShield may miss some internal stack behaviors that could indicate of attacks. In contrast, BlueSWAT and OASIS perform detection on the device itself. This allows them to collect more context information about the state of the connection, which is harder to gather as an external probe.

The systems differ significantly in how they are implemented and updated. OASIS requires extensive reverse engineering of firmwares to create patches, which are then used to

TABLE II
IDS COMPARISON OF BLE DETECTION APPROACHES

| Features | BlueShield [27] | OASIS [11] | BlueSWAT [12] | Sirocco |
|---|---|---|---|---|
| *Architecture* | Separate collectors and monitor | Patch the controller firmware ISR | Use eBPF programs | Monitoring thread in the controller |
| *Deployment* | External | On-device | On-device | On-device |
| *Network Type* | Stationary | Mobile & Stationary | Mobile & Stationary | Mobile & Stationary |
| *Attack Focus* | Spoofing | Spoofing, Signal Injection, MITM | Multi-layer | Spoofing, Signal Injection, MITM |
| *Detection Scope* | Advertising Phase | Advertising & Connection Phases | Connection Phase | Advertising & Connection Phases |
| *Detection Mechanisms* | Finite State Machines (FSM) | Features-based Heuristics | Features Anomalies | Features-based Heuristics |
| *Protocol State-aware* | No | Partial | Yes | Partial |
| *Implementation* | Non-intrusive | Binary Hooks | Binary Hooks | Source Code Patch |
| *Update* | Limited | Requires full firmware update | Over-The-Air via BLE | Requires full firmware update |
| *Stack Compatibility* | N/A | Broadcom, SoftDevice, Zephyr $< 3.0$ | NimBLE, SimpleLink, Zephyr, ESP-IDF, Bouffalo | Zephyr |

add detection capabilities. Instead, BlueSWAT needs a limited number of hooks in the controller and uses an eBPF framework to abstract the runtime instrumentation from the device's architecture and firmware code. This design lets BlueSWAT receive security policies over BLE and load them without restarting or changing the firmware. This approach makes updates easier and more flexible across different devices.

Current solutions mainly focus on alerting when they detect an attack, without offering immediate mitigations. Moreover, as explained by BlueSWAT's motivational example on the KNOB attack, it can be hard to tell the difference between legitimate use and attacks. This confusion raises concerns about false positives and confidence in the alerts issued. In addition, how a device can respond to an attack depends greatly on the role and the capabilities of the device. For example, a smart power outlet will have very different response options compared to a smartphone acting as a central device.

As we describe in the next section, `Sirocco` has been designed as a Host Intrusion Detection System that is capable of detecting various BLE attacks, targeting different layers of the BLE stack. It does not suffer from the drawbacks of the external probes and is based on the modification of the open-source Zephyr BLE stack. These choices make the implementation, integration and updates of BLE intrusion detection mechanisms easier than in other intrusion detection approaches: too specific (cover only specific attacks) or complex to integrate (binary patches).

## V. SIROCCO

### A. Design Choices

This section outlines the main design choices made during the development of `Sirocco`.

The design of `Sirocco` emphasizes flexibility, making it applicable for both central and peripheral roles. The roles of central and peripheral devices are intrinsically different. Central devices such as smartphones, manage multiple connections and have higher processing capabilities, while peripherals, typically sensors, operate under stricter resource constraints. Attacks also manifest differently depending on the role. For example, a central device might be more susceptible to MITM attacks, while a peripheral might be more prone to spoofing.

These main differences between central and peripheral roles present a challenge in designing defense mechanisms that are applicable to both. Peripheral devices are often fully implemented in a monolithic firmware that includes both the application and BLE stack. In contrast, central devices may use a dedicated firmware for the controller and rely on an Operating System such as Android or iOS, for the host stack. This variation in configurations complicates the design of a one-size-fits-all solution. As observed in OASIS [11], detection modules are often role-specific, implementing algorithms only for the central or peripheral role. We took a similar approach as the OASIS IDS with a core component providing the main structure where detection modules can be plugged to suit the best of the device needs in terms of detection.

One of the primary design challenges was to ensure system stability. This means that multiple detection modules could operate in parallel without disrupting active connections. The BLE timing constraints are strict, and any added overhead from detection algorithms could potentially interfere with ongoing communication. We observed such issues while experimenting with OASIS. Patching the Interrupt Service Routine (ISR) to include all detection algorithms leads to system instability and crashes. To address this, `Sirocco` minimizes the work performed within the ISR, limiting it to the collection of relevant packet metrics. These metrics are then transferred to a dedicated kernel thread that handles the execution of the corresponding detection modules. This choice improves system stability while allowing for real-time attack detection.

Although `Sirocco` focuses on attack detection, the design also covers the importance of response and remediation. Applications may need to adapt their behavior upon detecting suspicious activities. For this reason, `Sirocco` provides a callback mechanism when an alert is raised. For instance, a central application could terminate a connection when detecting a MITM attack. Moreover, nearby devices, such as sensors, external probes, or gateways, may benefit from being informed

about detected threats in their areas. In a spoofing attack, a peripheral application may broadcast advertisements to nearby devices for signaling an attack is underway. This inter-device communication allows for the transfer of alerts to a central system or Security Operations Center (SOC), facilitating a coordinated and effective response.

### B. Control flow

`Sirocco` extends the Zephyr RTOS [1] by integrating attack detection capabilities into its open-source Bluetooth Low Energy (BLE) stack. The Zephyr Project is an open-source real-time operating system (RTOS) designed for resource-constrained devices. It offers a highly modular and scalable platform with a large hardware support and implements various connectivity solutions, and in particular, its BLE stack is fully open-source and compliant with Bluetooth 5.3.

`Sirocco` has been implemented for the Nordic Semiconductor's nRF52840 System-on-Chips (SoC) and can be easily ported to other systems due to its hardware-agnostic design based on Zephyr RTOS abstractions. The system consists of a core component that aggregates metrics from both transmitted and received packets, captured directly in the Interrupt Service Routines (ISR). These metrics are then dispatched to corresponding detection modules. An alert-handling thread aggregates alerts from the detection algorithms, and provides notifications for applications and neighboring devices to enable coordinated security responses.

The control flow in `Sirocco` starts in the multiple BLE ISR. We inserted lightweight functions to gather various metrics for each packet. They differ depending on ISR function and the type of packet (e.g., `ADV_IND`, `SCAN_REQ`, `CONNECT_IND`). Relevant metrics include the packet timestamp, CRC validity, addresses, RSSI, or the channel. Metrics collection triggers on the following ISR: `adv_rx`, `adv_tx`, `scan_rx`, `scan_tx`, `conn_rx` and `conn_tx`. To minimize latency and maintain BLE timing requirements, the collected data is promptly transferred via a FIFO queue to the main thread. Moreover, the memory allocation mechanism is designed to optimize allocation speed in the ISR by deferring memory management operations to the less time-critical kernel thread during deallocation. Once processed in the main thread, metrics are dispatched to appropriate detection modules based on the device's role and the ISR event origin. Each detection module implements specific algorithms and stores essential data for continuity across multiple algorithm executions.

When suspicious activity is detected, an alert is raised and sent to the dedicated handling thread. It aggregates alerts, allowing for central management and processing of potential security incidents. A callback mechanism allows applications to adapt their behavior corresponding to the raised alerts. Alerts can be logged in the system, sent through a callback to the application, or broadcast to nearby devices as a special advertisement, depending on the developer's configuration.

### C. Detection modules

We have primarily ported and re-implemented the detection algorithms from OASIS [11] for the following five attacks: BTLEJuice [7], BTLEJack [8], KNOB [6], InjectaBLE [9] and GATTacker [18]. These algorithms analyze collected metrics for patterns or anomalies indicative of these known attacks.

The GATTacker detection module monitors advertisement channels from the central device to identify deviations in the expected channel hopping pattern. An alert is triggered if the estimated advertising interval is shorter than the detection threshold, indicating potential spoofed advertisements.

The BTLEJuice module operates on the peripheral device. Upon connection, a background scanning task checks advertisements for address matches with the device itself. An alert is raised if a match is found, indicating potential spoofing. Let us note that the need for background scanning induces a heightened power consumption. However, to our knowledge, there are only two options for detecting spoofing in this setup. One can either detect that someone that should not be emitting is with background scanning, or authenticate the source of a message. This second method requires an authentication scheme, which is present but scarcely used, or a physical-layer based identification, which is hard to embed on BLE chips without modifications as it requires the chip's controller to access raw signal.

The original BTLEJack detection module is implemented on the central device. It monitors consecutive packets with invalid CRCs during connection events. Because this situation is unlikely under normal conditions, an alert is raised if the number of corrupted packets exceeds a pre-defined threshold. However, this module only works for corrupted packets following the same processing chain as the normal ones. We observed that, in our case, the jamming was corrupting the preamble, such that the receiver couldn't synchronize on the packets. This makes it appear as if the connection event doesn't contain any packet from the peripheral. We then need to also treat that case as a potential packet corruption, at the cost of raising false positives when a large number of connection events are legitimately empty. This is notably the case when the peripheral disconnects improperly.

The InjectaBLE detection module functions on the peripheral device. It monitors the timing between received packets for irregular intervals. An alert is raised if the interval is shorter than expected, suggesting potential packet injection. The idea behind this is that the slave synchronizes with the attacker, which is supposed to have sent its packet earlier, making the slave answer earlier too. However, the drawback of this metric lies in the fact that successful attackers must inject packets precisely within the legitimate reception window, making anomalous timing difficult to distinguish from normal variations as it could be legitimate. Indeed, when trying to reproduce the results from OASIS, the attacker we used generated timings higher than the earliest legitimate one. Thus, setting a threshold to detect such attacks would always generate a large number of false positives. We suspect this metric could detect attacks that fail by trying to send before the window, or that reach its very beginning. The first case could correspond to an unsuccessful attacker with a low-quality window estimate, while the second seems more unrealistic.

The KNOB detection module is applicable to both central and peripheral devices. It passively monitors the entropy value

in Pairing Requests. An alert is triggered if the entropy value is less than 10 bytes, indicating a vulnerability to brute-force.

## VI. Evaluation

### A. Experiments Setup

We evaluated the `Sirocco` approach with eight examples, with different BLE roles, from Zephyr 3.5.99 and NRF Connect SDK v2.6.0. All codes produced during the experiment as well as the experiment results are available at the following address: https://github.com/plrolivier/sirocco-examples

- Zephyr BLE Central Heart Rate Monitor (*C1*)
- nRF Connect BLE Central BAS (*C2*)
- nRF Connect BLE Central HIDS (*C3*)
- Zephyr BLE Peripheral Heart Rate Monitor (*P1*)
- nRF Connect BLE Peripheral HIDS keyboard (*P2*)
- nRF Connect BLE Peripheral HIDS mouse (*P3*)
- Zephyr BLE HCI USB (*CP1*)
- Zephyr BLE HCI UART (*CP2*)

For hardware, we tested our code for the nRF52840 SoC on nRF52840-DK and nRF52840-Dongle boards. The nRF52840 SoC is based on a 32 bits ARM Cortex M4 CPU running at 64 MHz with 1 MB of Flash memory and 256 KB of RAM.

### B. Attack detection

We use the Mirage [10] framework to launch attack campaigns. Table III presents the detection rates for the evaluated attacks. We conducted 1000 runs randomly alternating between attacks and benign traffic for both BTLEJuice and KNOB attacks. Before counting a detection as a true positive, we verify that the attack was successfully executed. During our tests, we observed 473 successful attacks launched for BTLEJuice and 482 for KNOB. All successful attacks launched were successfully detected, and no false alerts were triggered during benign traffic. The total number of true positives and true negatives does not sum to 1000 because we excluded failed attack attempts from our analysis, focusing only on successful attack executions and legitimate connections.

We then ran 200 connections running a full GATT discovery on the peripheral, alternating between normal connections and attacks with BLTEJack. All 100 attacks were detected by `Sirocco`. Over the 100 benign connections, four alerts were raised. Moreover, the detection metric could also raise alerts in legitimate connections in the case of a device leaving the connection without disconnecting with the proper procedure.

In the case of GATTacker, the attack takes place before any connection, meaning benign traffic is harder to define. In this experiment, we first let the central and peripheral run for one hour without attacks, rebooting the central every 45 seconds. This results in 80 benign periods. During this part, no alert was raised by `Sirocco`. Then, we repeated the attack 100 times, the attacker starting to emit after the central had learned the legitimate inter-advertisement timing. This time, all attacks were successfully detected. For this experiment, the peripheral was selecting its advertising interval between 125 and 156.25 ms, while the attacker was using a fixed value of 12.5 ms.

The significant difference in the number of experiments can be attributed to the difficulty in automating certain attacks.

Due to the drawbacks of the InjectaBLE detection heuristic, we were unable to reproduce the OASIS detection results. Thus, we do not present results related to this attack in Table III. However, we still ported the original detection module to our framework to evaluate the performance impact in the following sections.

### C. Latency Overhead

We evaluated the latency overhead introduced by `Sirocco` through a comprehensive analysis of three key components: end-to-end system latency, ISR and FIFO latencies. Our methodology focused on the `conn_rx` and `scan_rx` ISRs, as these represent the primary sources of detection metrics. All time measurements were made using the CPU cycle counter on an nRF52840-DK development board. When possible, we enabled scenario-specific detection modules: KNOB and BTLJack modules for Central HR (C1) on `conn_rx`, GATTacker on `scan_rx`, and KNOB, BTLEJuice and InjectaBLE for Peripheral HR (C1) on `conn_rx`, We ran an additional example called Observer with `scan_rx` and GATTacker module.

*1) End-to-end time overhead:* We measured the total system latency from metrics collection in the ISR to processing by the detection algorithms. Our experiments ran for five minutes on both `conn_rx` and `scan_rx` ISR, with their compatible detection modules enabled.

For `conn_rx`, we observed average latencies of 20 $\mu$s and 9 $\mu$s for Central HR (C1) and Peripheral HR (P1) scenarios, with maximum latencies reaching 32 $\mu$s in the worst case. The `scan_rx` path showed higher latencies, averaging 47 $\mu$s with maximum peaks of 129 $\mu$s. Despite these peaks, the system maintained stable operation throughout the test period.

*2) ISR Latency:* To better understand the overhead source, we measured the time spent in modified ISR.

The results revealed that our metric collection introduced a significant but manageable overhead in the ISRs. The `conn_rx` routine experienced a 143% and 131% rise compared to the baseline. However, the impact on the `scan_rx` routine was more moderate with a 16.1% and 29.6% increases. More important, our worst-case ISR execution time of 10,3 $\mu$s stays well below the 150 $\mu$s needed for BLE. While there may be room for further optimization in our metric collection, these encouraging results demonstrate the benefits of our approach.

*3) FIFO Throughput and Latency:* The final component of our analysis focused on the FIFO transfer time between the ISR and the lower-priority kernel thread. During five-minute test runs, we monitored the latency between when the metrics are inserted in the FIFO to when they are received.

The FIFO component showed average latencies of 8-16$\mu$s for `conn_rx` scenarios and 17-45$\mu$s for `scan_rx` scenarios. Maximum FIFO latencies peaked at 31$\mu$s for `conn_rx` and 82$\mu$s for `scan_rx`.

Comparing these results with the end-to-end measurements reveals that FIFO operations constitute a significant portion of the total latency. For instance, in the Central HR (C1) scenario, the FIFO latency (16 $\mu$s average) accounts for approximately 80 % of the end-to-end latency (20 $\mu$s average). This suggests that FIFO operations represent the primary bottleneck in our system, rather than the ISR metric collection itself.

TABLE III
DETECTION RATE.

| Experiment | Target | TP | FP | TN | FN | Recall | Precision |
|---|---|---|---|---|---|---|---|
| **GATTacker** | Central HR (C1) | 100 | 0 | 80 | 0 | 1.0 | 1.0 |
| **BTLEJuice** | Peripheral HR (P1) | 473 | 0 | 513 | 0 | 1.0 | 1.0 |
| **BTLEJack** | Central HR (C1) | 100 | 4 | 96 | 0 | 1.0 | 0.96 |
| **KNOB** | Peripheral HR (P1) | 482 | 0 | 480 | 0 | 1.0 | 1.0 |

TABLE IV
END-TO-END LATENCY.

| | | Total packets processed | Average CPU cycles | Average $\mu$s | Minimum CPU cycles | Minimum $\mu$s | Maximum CPU cycles | Maximum $\mu$s |
|---|---|---|---|---|---|---|---|---|
| `conn_rx` | Central HR (C1) | 6000 | 1312 | **20** | 1312 | 20 | 1319 | 20 |
| | Peripheral HR (P1) | 6000 | 599 | **9** | 571 | 8 | 2070 | 32 |
| `scan_rx` | Central HR (C1) | 612 | 2973 | **46** | 1398 | 21 | 6594 | 103 |
| | Observer | 973 | 3132 | **48** | 1734 | 27 | 8309 | 129 |

TABLE V
ISR OVERHEAD LATENCY INTRODUCED BY METRICS COLLECTION.

| | | Sirocco disabled CPU cycles | Sirocco disabled $\mu$s | Sirocco enabled CPU cycles | Sirocco enabled $\mu$s | Increase % |
|---|---|---|---|---|---|---|
| `conn_rx` | Central HR (C1) | 230 | 3,6 | 559 | 8,7 | **143** |
| | Peripheral HR (P1) | 286 | 4,5 | 660 | **10,3** | **130,8** |
| `scan_rx` | Central HR (C1) | 311 | 4,9 | 361 | 5,7 | **16,1** |
| | Observer | 253 | 4,0 | 328 | 5,1 | **29,6** |

TABLE VI
FIFO LATENCY.

| | | Total packets processed | Average CPU cycles | Average $\mu$s | Minimum CPU cycles | Minimum $\mu$s | Maximum CPU cycles | Maximum $\mu$s |
|---|---|---|---|---|---|---|---|---|
| `conn_rx` | Central HR (C1) | 5993 | 1080 | **16** | 546 | 8 | 1089 | 17 |
| | Peripheral HR (P1) | 6000 | 550 | **8** | 527 | 8 | 1989 | 31 |
| `scan_rx` | Central HR (C1) | 2148 | 2888 | **45** | 620 | 9 | 5256 | 82 |
| | Observer | 1095 | 1113 | **17** | 552 | 8 | 4257 | 66 |

Overall, while our implementation introduces measurable overhead, it maintains reasonable latency bounds that should not impact normal BLE operation. The total system latency remains well within acceptable limits for real-time detection, with the majority of overhead attributable to FIFO operations rather than metric collection.

### *D. Memory usage*

We conducted an analysis of the memory footprint of the `Sirocco` framework, focusing on both flash and RAM usage across a selected set of examples. To measure the memory footprint, we utilized the built-in compiler, which extracts these measurements from the produced Zephyr ELF image. Table VII presents our findings for flash memory utilization.

Our results show that the core `Sirocco` framework introduces an additional memory overhead ranging from 5,296 to 6,260 bytes. This increment represents approximately 3% of the total flash memory usage, which we consider a modest increase given the framework's capabilities. The majority of detection modules contribute only minimal additional memory requirements, typically around a few hundred bytes. However, two modules stand out as exceptions. The BTLEJuice module exhibits a significantly larger memory footprint. This increased size is attributed to its dependency on the `CONFIG_BT_OBSERVER` setting, which is necessary to enable scanning capabilities while maintaining a connection. The inclusion of this functionality results in an additional code size of approximately 8,000 bytes. The GATTacker module requires the `CONFIG_RING_BUFFER` feature, which also contributes to a slight increase in memory usage.

RAM utilization in our framework is heavily dependent on the `CONFIG_HEAP_MEM_POOL_SIZE` parameter, which sizes the FIFO buffers between the ISR and analysis thread. During high Bluetooth traffic, insufficient heap space can lead to packet drops in these FIFOs. This reliance on heap size for buffer management is a limitation of our current approach.

### *E. Power Consumption*

We used the Power Profiler Kit 2 (PPK2) from Nordic Semiconductor to measure power consumption during both advertising and connection phases. Measurements were conducted over a five-minute run.

Results are presented in table VIII. The Peripheral HR (P1) exhibited a notable 13.67% increase in power consumption. This significant rise can be attributed to the BTLEJuice module's requirement of `CONFIG_OBSERVER`, which maintains background scanning during the connection. When the BTLEJuice module was disabled, power consumption marginally

TABLE VII
FLASH MEMORY USAGE.

| Examples | | Plain | Sirocco without modules | With BTLEJack | With BTLEJuice | With GATTacker | With InjectaBLE | With KNOB | With all modules |
|---|---|---|---|---|---|---|---|---|---|
| Central HR (C1) | Size (B) | 181016 | 186548 | 187188 | NA | 187652 | NA | 187036 | 188140 |
| | Diff (B) | — | 5532 | 6172 | — | 6636 | — | 6020 | 7124 |
| | Diff (%) | — | 2.97% | 3.30% | — | 3.54% | — | 3.22% | **3.79%** |
| Central BAS (C2) | Size (B) | 208212 | 214424 | 215236 | NA | 215876 | NA | 215092 | 216348 |
| | Diff (B) | — | 6212 | 7024 | — | 7664 | — | 6880 | 8136 |
| | Diff (%) | — | 2.90% | 3.26% | — | 3.55% | — | 3.20% | **3.76%** |
| Central HIDS (C3) | Size (B) | 213484 | 219692 | 220508 | NA | 221144 | NA | 220360 | 221620 |
| | Diff (B) | — | 6208 | 7024 | — | 7660 | — | 6876 | 8136 |
| | Diff (%) | — | 2.83% | 3.19% | — | 3.46% | — | 3.12% | **3.67%** |
| Peripheral HR (P1) | Size (B) | 176148 | 181444 | NA | 189696 | 182560 | 182916 | 181932 | 193500 |
| | Diff (B) | — | 5296 | — | 13548 | 6412 | 6768 | 5784 | 17352 |
| | Diff (%) | — | 2.92% | — | 7.14% | 3.51% | 3.70% | 3.18% | **8.97%** |
| Peripheral HIDS Keyboard (P2) | Size (B) | 217808 | 223104 | NA | 230944 | 223796 | 224424 | 223452 | 232764 |
| | Diff (B) | — | 5296 | — | 13136 | 5988 | 6616 | 5644 | 14956 |
| | Diff (%) | — | 2.37% | — | 5.69% | 2.68% | 2.95% | 2.53% | **6.43%** |
| Peripheral HIDS Mouse (P3) | Size (B) | 209676 | 215568 | NA | 223176 | 216604 | 216812 | 215836 | 225436 |
| | Diff (B) | — | 5892 | — | 13500 | 6928 | 7136 | 6160 | 15760 |
| | Diff (%) | — | 2.73% | — | 6.05% | 3.20% | 3.29% | 2.85% | **6.99%** |
| HCI USB (CP1) | Size (B) | 152420 | 158108 | 158764 | NA | 159228 | 159580 | 158596 | 160836 |
| | Diff (B) | — | 5688 | 6344 | — | 6808 | 7160 | 6176 | 8416 |
| | Diff (%) | — | 3.60% | 4.00% | — | 4.28% | 4.49% | 3.89% | **5.23%** |
| HCI UART (CP2) | Size (B) | 143892 | 150152 | 150948 | NA | 152068 | 151608 | 150640 | 153816 |
| | Diff (B) | — | 6260 | 7056 | — | 8176 | 7716 | 6748 | 9924 |
| | Diff (%) | — | 4.17% | 4.67% | — | 5.38% | 5.09% | 4.48% | **6.45%** |

TABLE VIII
POWER CONSUMPTION OVER A FIVE MINUTE RUN.

| Example | Sirocco Status | Power (mW) Mean | Power (mW) Std dev | Increase (%) |
|---|---|---|---|---|
| Central HR (C1) *connection* | Off | 130,70 | 0,86 | — |
| | On | 130,82 | 0,97 | **0,09** |
| Peripheral HR (P1) *connection* | Off | 141,97 | 11,28 | — |
| | On | 161,38 | 11,44 | **13,67** |
| | On[1] | 142,22 | 11,30 | **0,18** |
| Observer *advertising* | Off | 127,32 | 0,58 | — |
| | On | 127,39 | 0,58 | **0,05** |
| Peripheral HR (P1) *advertising* | Off | 119,32 | 11,02 | — |
| | On | 119,85 | 11,03 | **0,44** |

increased by 0.18% from the baseline. Other configurations demonstrated minimal power consumption variations, ranging from 0.05% to 0.44%. The high standard deviation for Peripheral HR (P1) stems from the firmware's alternating behavior between BLE data transmission and sleep states.

## VII. DISCUSSION

The architecture or `Sirocco`, which decouples the collection of metrics in the ISR from the analysis in a dedicated thread, allows for high stability, as it does not temper excessively with the communication timings. However, this approach incurs a higher RAM usage due to the need for inter-thread communication buffers. The increase in memory footprint, while contributing to system reliability, presents a trade-off between performance and resource utilization. Future optimizations could focus on minimizing this RAM overhead without compromising detection capabilities and stability.

**Alert handling** is crucial for intrusion detection. `Sirocco`'s current focus on alert generation provides a foundation for security measures. Logging these alerts facilitates forensic analysis and long-term security improvements. However, real-time response mechanisms warrant further investigation. Immediate actions, such as connection termination or security parameter adjustment, enhance protection but risk impacting user experience if not carefully implemented. For instance, imposing connection timeouts after detecting spoofing attempts may inadvertently cause denial of service. The concept of sharing alerts with nearby devices or a central management system offers potential for collaborative threat detection.

Among the attacks tested, we were unable to reproduce the results from OASIS on InjectaBLE because of the imprecision of the detection heuristic. We are currently working on alternative detection methods for this attack, based on discrepancies in the distribution of reception timings. Indeed, when an attacker is present, we should observe a shift of the timing distribution towards smaller inter-frame intervals. However, an heuristic solely based on this distribution still has difficulty detecting an attacker succeeding in a few attempts, or only injecting a small number of frames. Note that this attack might still be detected by application-level measures, which are out of the scope of this article.

## VIII. CONCLUSION

In this paper, we presented `Sirocco`, our novel host-based intrusion detection system based on the Zephyr RTOS. This system leverages the information available at the firmware level to detect multiple Bluetooth Low Energy attacks with high detection accuracy and efficiency.

Looking back at our initial objectives, `Sirocco` successfully delivers on its goals. First, our framework demonstrates integration with existing device applications and adapting to specificities to both central and peripheral BLE device roles.

Second, `Sirocco` achieves real-time attack detection with customizable response mechanisms. The alert-handling thread enables applications to adapt their behavior when suspicious activities are detected, while also providing notification capabilities for neighboring devices. Finally, our separation of metric collection and analysis processes minimizes performance overhead while maintaining system stability. Experimental evaluation confirms that `Sirocco` introduces minimal impact on power consumption (as low as 0.05% in some configurations) and reasonable latency overheads that don't affect normal device operation. While the BTLEJuice module does increase power consumption by 13.67% due to background scanning, this only affects peripheral role.

This approach, being based on the open-source code of Zephyr, is easier to adapt to new attacks than approaches requiring direct firmware modifications. Moreover, Sirocco's architecture leverages Zephyr's wide range of supported devices, making it a practical solution for improving BLE security across diverse hardware implementations.

## Acknowledgement

This work has been partially supported by the French National Research Agency under the France 2030 label (Superviz ANR-22-PECY-0008). The views reflected herein do not necessarily reflect the opinion of the French government.

## References

[1] Zephyr RTOS Project. URL: https://zephyrproject.org/.

[2] ANSSI. Pairing Mode Confusion in BLE Passkey Entry. URL: https://www.bluetooth.com/learn-about-bluetooth/key-attributes/bluetooth-security/confusion-in-ble-passkey/.

[3] Daniele Antonioli, Nils Ole Tippenhauer, and Kasper Rasmussen. Bias: Bluetooth impersonation attacks. In *2020 IEEE symposium on security and privacy (SP)*, 2020.

[4] Daniele Antonioli, Nils Ole Tippenhauer, and Kasper Rasmussen. Key negotiation downgrade attacks on bluetooth and bluetooth low energy. *ACM Transactions on Privacy and Security (TOPS)*, 2020.

[5] Daniele Antonioli, Nils Ole Tippenhauer, Kasper Rasmussen, and Mathias Payer. Blurtooth: Exploiting cross-transport key derivation in bluetooth classic and bluetooth low energy. In *Proceedings of the 2022 ACM on Asia conference on computer and communications security*, 2022.

[6] Daniele Antonioli, Nils Ole Tippenhauer, and Kasper B Rasmussen. The KNOB is broken: Exploiting low entropy in the encryption key negotiation of bluetooth BR/EDR. In *28th USENIX security symposium (USENIX security 19)*, 2019.

[7] Damien Cauquil. Btlejuice: The bluetooth Smart MitM Framework, 2016.

[8] Damien Cauquil. You'd better secure your BLE devices or we'll kick your butts !, 2018.

[9] Romain Cayre, Florent Galtier, Guillaume Auriol, Vincent Nicomette, Mohamed Kaâniche, and Géraldine Marconato. Injectable: Injecting malicious traffic into established bluetooth low energy connections. In *51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2021.

[10] Romain Cayre, Vincent Nicomette, Guillaume Auriol, Eric Alata, Mohamed Kaaniche, and Geraldine Marconato. Mirage: towards a metasploit-like framework for iot. In *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*, 2019.

[11] Romain Cayre, Vincent Nicomette, Guillaume Auriol, Mohamed Kaâniche, and Aurélien 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*, 2024.

[12] Xijia Che, Yi He, Xuewei Feng, Kun Sun, Ke Xu, and Qi Li. Blueswat: A lightweight state-aware security framework for bluetooth low energy. 2024.

[13] Jiska Classen, Francesco Gringoli, Michael Hermann, and Matthias Hollick. Attacks on wireless coexistence: Exploiting cross-technology performance features for inter-chip privilege escalation. In *2022 IEEE Symposium on Security and Privacy (SP)*, 2022.

[14] Jiska Classen and Matthias Hollick. Happy mitm: fun and toys in every bluetooth device. In *Proceedings of the 14th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, 2021.

[15] Tristan Claverie and José Lopes Esteves. Bluemirror: reflections on bluetooth pairing and provisioning protocols. In *2021 IEEE Security and Privacy Workshops (SPW)*, 2021.

[16] Matheus E Garbelini, Vaibhav Bedi, Sudipta Chattopadhyay, Sumei Sun, and Ernest Kurniawan. BrakTooth: Causing havoc on bluetooth link manager via directed fuzzing. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 1025–1042, 2022.

[17] Matheus E Garbelini, Chundong Wang, Sudipta Chattopadhyay, Sun Sumei, and Ernest Kurniawan. SweynTooth: unleashing mayhem over bluetooth low energy. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, 2020.

[18] Sławomir Jasek. Gattacking Bluetooth Smart Devices, 2016.

[19] Imtiaz Karim, Abdullah Al Ishtiaq, Syed Rafiul Hussain, and Elisa Bertino. Blediff: Scalable and property-agnostic noncompliance checking for ble implementations. In *2023 IEEE Symposium on Security and Privacy (SP)*, 2023.

[20] Nordic Semiconductor. Legacy pairing vs LE Secure Connections. URL: https://academy.nordicsemi.com/topic/legacy-pairing-vs-le-secure-connections-2/.

[21] Jan Ruge, Jiska Classen, Francesco Gringoli, and Matthias Hollick. Frankenstein: Advanced wireless fuzzing to exploit new bluetooth escalation targets. In *29th USENIX Security Symposium (USENIX Security 20)*, 2020.

[22] Mike Ryan. Bluetooth: With low energy comes low security. In *7th USENIX Workshop on Offensive Technologies (WOOT 13)*, 2013.

[23] Min Shi, Jing Chen, Kun He, Haoran Zhao, Meng Jia, and Ruiying Du. Formal analysis and patching of BLE-SC pairing. In *32nd USENIX Security Symposium (USENIX Security 23)*, 2023.

[24] Bluetooth SIG. Bluetooth Core Specification v5.4. URL: https://www.bluetooth.com/specifications/specs/core-specification-5-4/.

[25] Jiliang Wang, Feng Hu, Ye Zhou, Yunhao Liu, Hanyi Zhang, and Zhe Liu. Bluedoor: breaking the secure information flow via ble vulnerability. In *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*, 2020.

[26] Haohuang Wen, Zhiqiang Lin, and Yinqian Zhang. Firmxray: Detecting bluetooth link layer vulnerabilities from bare-metal firmware. In *Proceedings of the 2020 ACM SIGSAC conference on computer and communications security*, 2020.

[27] Jianliang Wu, Yuhong Nan, Vireshwar Kumar, Mathias Payer, and Dongyan Xu. BlueShield: Detecting spoofing attacks in bluetooth low energy networks. In *23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020)*, 2020.

[28] Jianliang Wu, Yuhong Nan, Vireshwar Kumar, Dave Jing Tian, Antonio Bianchi, Mathias Payer, and Dongyan Xu. BLESA: Spoofing attacks against reconnections in bluetooth low energy. In *14th USENIX Workshop on Offensive Technologies (WOOT 20)*, 2020.

[29] Jianliang Wu, Ruoyu Wu, Dongyan Xu, Dave Tian, and Antonio Bianchi. Sok: The long journey of exploiting and defending the legacy of king harald bluetooth. In *2024 IEEE Symposium on Security and Privacy (SP)*, 2023.

[30] Jianliang Wu, Ruoyu Wu, Dongyan Xu, Dave Jing Tian, and Antonio Bianchi. Formal model-driven discovery of bluetooth protocol design vulnerabilities. In *2022 IEEE Symposium on Security and Privacy (SP)*, 2022.