VaktBLE: A Benevolent Man-in-the-Middle Bridge to Guard against Malevolent BLE Connections

Geovani Benita
SUTD

Leonardo Sestrem
SUTD

Matheus E. Garbelini
SUTD

Sudipta Chattopadhyay
SUTD

Sumei Sun
*A*STAR I2R*

Ernest Kurniawan
*A*STAR I2R*

***Abstract*—In this paper, we conceptualize, design and evaluate *VaktBLE*, a novel framework to defend BLE peripherals against low-level BLE attacks. *VaktBLE* presents a novel, efficient and (almost) deterministic technique to silently hijack the connection between a potentially malicious BLE central and the target peripheral to be protected. This creates a benevolent man-in-the-middle (MiTM) bridge that allows us to validate each packet sent by the BLE central. For validation, we implement a flexible and extensible framework to detect a variety of attacks due to packets that are invalid, out-of-order or flooded. An appealing capability of *VaktBLE* is that it can validate all packets down to the link layer, thus allowing us to defend against complex BLE attacks that bypass state-of-the art binary patching frameworks. We have implemented *VaktBLE* and evaluated it with 25 state-of-the-art BLE attack vectors from offensive tools such as SweynTooth, CyRC and BLEDiff. Our evaluation shows that *VaktBLE* effectively detects all these attacks and the *VaktBLE* MitM bridge incurs only $10ms$ overhead. Moreover, we have evaluated the capability and robustness of *VaktBLE* against several adaptive attacks including fuzzing-based attacks. We also show the extensibility of *VaktBLE* to counteract protocol-level attacks and rogue peripherals. Our evaluation reveals that *VaktBLE* not only stops fuzzing-based attacks with high effectiveness (97.5%), but *VaktBLE* also does not incur false positives when attacks are randomly mixed with benign connection attempts.**

1. Introduction

Bluetooth Low Energy (BLE) is a widely used wireless communication protocol for internet-of-things (IoT) devices. Past few years have seen an increasing number of BLE implementation vulnerabilities [1], [2], [3], many of which target the closed-source Link Layer (LL) of BLE protocol. Unfortunately, patches for such vulnerabilities may often take a significant time to reach the end users (e.g., IoT users), if at all. Hence, it is desirable to protect IoT devices from attacks that may exploit hidden and unpatched vulnerabilities in devices' BLE implementation.

In this paper, we propose *VaktBLE*, a novel approach to defend *existing peripheral* against the recent rise on BLE offensive tools e.g., CyRC, Sweyntooth, BLEDiff and others [4], [2], [3], [5]. *VaktBLE is the fist real-time, hardware-software platform that guards the target peripheral from such low-level protocol vulnerabilities in a practical over-the-air fashion* (as depicted in Figure 1). To the best of our knowledge, *VaktBLE* is also the first system to provide protection for BLE peripherals without requiring patching. This makes *VaktBLE* practical for home/enterprise environments with fewer security-sensitive devices e.g., door locks, smart power outlets, medical equipment, etc. Furthermore, the open platform architecture offered by *VaktBLE* facilitates further research and development in areas such as BLE jamming mitigation and real-time intrusion prevention systems for BLE devices.

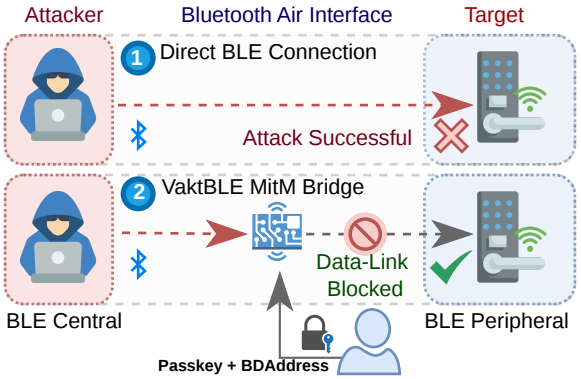

Figure 1. An Illustration of *VaktBLE* protecting the target (BLE Peripheral) against the attacker (BLE Central).

VaktBLE requires no specialized expertise and can be easily deployed by non-expert users, securing low-level protocols such as Link Layer. This allows the user to temporarily *guard* BLE peripheral, which might take significantly long time to receive patches (if at all). *VaktBLE* can be easily deployed out-of-the-box to protect arbitrary BLE peripheral. As shown in Figure 1, the user simply needs to position *VaktBLE* physically close to the target peripheral and input the peripheral's *BDAddress* (and passkey, if required) into *VaktBLE*'s configuration. Unlike *VaktBLE*, state-of-the-art solutions demand expertise in firmware programming, reverse engineering, and binary patching. While VaktBLE's current implementation may be unsuitable to defend thousands of devices, we note that our jamming strategy is

generic and could be employed in software defined radios supporting many connections.

VaktBLE jams the central packet that is transmitted during *the start of a BLE connection* (see Figure 3), thus causing the target peripheral to never receive such packet due to CRC error. Then, *VaktBLE* takes control of the communication with the central by simply proceeding with the connection as per Bluetooth Core Specification [6]. Consequently, our technique of hijacking the communication with the central (i.e., peripheral impersonation) is simpler and unique from prior works which apply jamming during either the *Discovery* phase or after the connection has already been established (see Figure 3). Specifically, prior works require jamming and spoofing many packets in the *Discovery* phase [7], [8] (i.e., peripheral advertisements). Similarly, works that hijack an established BLE connection [1], [9] are not applicable to our over-the-air defense since it takes time to hijack the connection, thus allowing the central to attack the peripheral in the meantime. Moreover, *VaktBLE* addresses the challenge of high delay when switching from radio reception to transmission as reported in prior works [9]. This is achieved by using an undocumented feature of the *nRF52840 Radio* v1.7 (as described in product specification Section 6.20.5).

Prior works on BLE defense use commercial Bluetooth dongles which do not offer control of the BLE Link Layer to the MitM bridge [10]. Additionally, LightBlue [11] involves user expertise and access to firmware binaries for patching and fails to protect against several BLE Link-layer attacks. In contrast, *VaktBLE* can be used completely out-of-the-box, yet it protects BLE peripherals down to link layer.

After providing a brief overview of *VaktBLE* (Section 2), we make the following contributions:

1) We present a novel jamming technique to silently hijack the connection (Section 3).
2) We present our comprehensive and extensible software validation component that checks attacks due to invalid packets, out-of-order packets and flooding (Section 3). Indeed, based on official data from 2019 to 2024 [12], we analyzed that these three attack scenarios cover about 54% (49/90) CVEs related to BLE vulnerabilities.
3) We implement and evaluate *VaktBLE* with 25 state-of-the-art attacks from tools e.g., Sweyntooth [2], BLEDiff [3], CyRC [4] and InjectaBLE [1]. Our evaluation reveals that *VaktBLE* effectively and efficiently stops all these attacks in various configurations (Sections 4-5).
4) We show the efficiency of *VaktBLE* in terms of the timing overhead introduced in the bridge. We show that the median latency is only $10ms$, significantly less to the minimum connection interval as per the Bluetooth core specification (Section 5).
5) We show that *VaktBLE* detects attacks within 2cm attacker distance and even if the attacker is 10m away, *VaktBLE* detects 100% of attack attempts (Section 5).

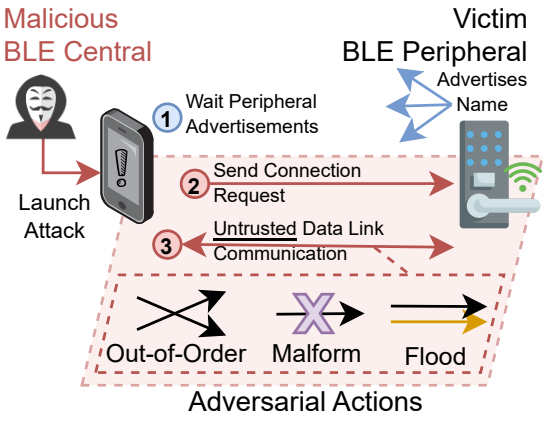

Figure 2. Illustration of attack model considered by *VaktBLE* during the BLE connection procedure. In such scenario, the malicious BLE central can inject any malicious packet during the BLE connection procedure (steps 2 and 3).

6) We evaluate *VaktBLE* with several adaptive attacks, including potential attacks launched by a state-of-the-art fuzzer such as Sweyntooth [2]. In an one-hour fuzzing session, our evaluation reveals even in the presence of potentially unknown attack attempts launched by the fuzzer, *VaktBLE* stops 97.5% (1170/1200) of these attempts. In our evaluation with potential attacks and benign connection, *VaktBLE* also does not exhibit any false positives (Section 5).
7) We compare *VaktBLE* with LightBlue [11] and show that the link-layer attacks stopped by *VaktBLE* can bypass protections offered by such debloater (Section 5).

Aside from these contributions, we foresee *VaktBLE* as an essential tool in the arsenal of wireless security researchers and as a strong foundation for further studies on BLE intrusion prevention and jamming techniques. We discuss some limitations and future enhancements of *VaktBLE* in Section 6 before concluding in Section 8.

2. Overview

Attacker Model: The attacker model scenario that *VaktBLE* aims to defend against is depicted in Figure 2, which illustrates a normal BLE connection procedure between the central and periphral. In this context, the adversary is represented as a malicious BLE Central (e.g., a smartphone), that initially waits for the victim (i.e., BLE peripheral, smartlock) advertisements in step ❶. Subsequently, the attacker sends a connection request in step ❷ and establishes direct control over the *Untrusted* Data Link communication in step ❸. This sequence of steps is inherent to bluetooth standard and such can always be performed between a *connectable* peripheral and an arbitrary central. Consequently, this means that the attacker has the capability to perform certain *Adversarial Actions* during communication such as replay, injection, flooding, drop, and transmission of out-of-order or malformed packets in an attempt to trigger vulnerabilities

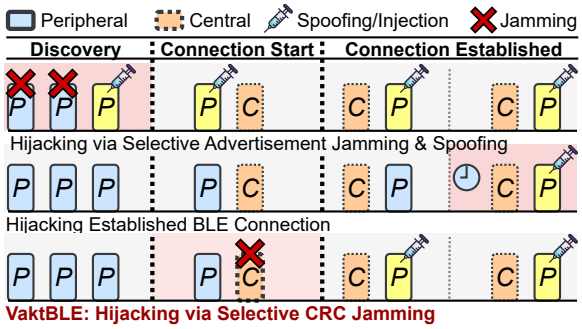

Figure 3. Simplified Illustration of *VaktBLE* Novel BLE Jamming Technique when compared to prior works. ***C:*** packets sent from *Central*. ***P:*** packets sent from *Peripheral*.

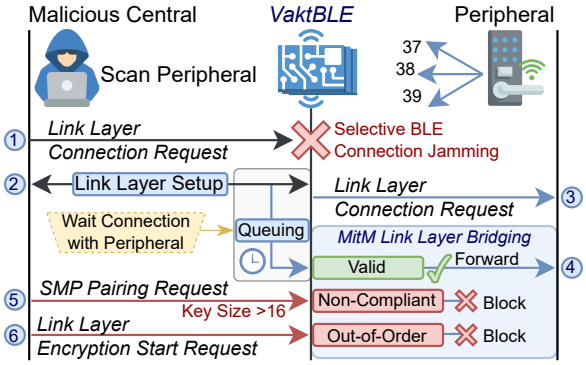

Figure 4. An attacker attempting to trigger *Key Size Overflow* against a peripheral under protection of *VaktBLE*.

within the victim (i.e., peripheral) without ever reaching authentication procedures of BLE. Given the potential for such attacks, any packet originating from the central device is treated as inherently *untrusted*. Consequently, certain validation procedures ought to take place such that only legitimate and non-malicious data can be forwarded to the peripheral, thereby heavily mitigating the risk of successful exploitation by a malicious central.

Key Insight: Broadly, *VaktBLE* involves two key procedures. Firstly, it introduces a novel technique to silently hijack the connection between the central and the target peripheral. This, creates a man-in-the-middle (MitM) bridge between the central and the target peripheral. Secondly, *VaktBLE* employs a validation component that forwards packets sent by the central to the target peripheral only if the packets are legitimate. To the best of our knowledge, *VaktBLE is the first work to (i) leverage a BLE MitM bridge for defense instead of performing attacks, and (ii) to enable* real-time BLE link-layer defense *instead of just intrusion detection*.

VaktBLE is a hardware and software framework that enables the user to defend an arbitrary BLE peripheral against BLE attacks exploiting known and unknown BLE vulnerabilities. To take advantage of *VaktBLE*, the user deploys the hardware running *VaktBLE* nearby the BLE peripheral to be protected. Figure 1 shows a common scenario (1) where the attacker directly connects to a target peripheral (e.g., BLE Smart Lock) and launches attacks over the BLE data link. BLE peripherals commonly accept connections from arbitrary centrals and exchange protocol messages before authentication. Hence, a peripheral with vulnerabilities in its BLE protocol implementation can be exploited during early communication with an attacker-controlled central [2]. Figure 1 (2) illustrates a novel real-time and non-intrusive countermeasure against BLE protocol (implementation) vulnerabilities by introducing *VaktBLE* near the peripheral. *VaktBLE* listens for connection attempts towards the target peripheral. Upon detecting a Connection Indication from an arbitrary BLE central it hijacks the Link Layer connection between the central and peripheral. *VaktBLE* then acts as a friendly Link Layer MitM bridge, validating packets received from the central before forwarding to the peripheral. If the central launches an attack, *VaktBLE* invalidates and blocks associated malformed, out-of-order, or flooding packets, preventing exploitation attempts from reaching the legitimate peripheral.

VaktBLE exerts its defense in a completely non-intrusive fashion, not demanding any modification or re-configuration of the legitimate peripheral. Instead, *VaktBLE* hijacks BLE connection attempts by tracking the peripheral's advertisement address within a Connection Indication packet. Consequently, the hijacking and establishment of the friendly MitM bridge is performed completely over-the-air, enabling easy deployment of *VaktBLE* to protect arbitrary BLE peripherals.

Benevolent MitM Bridge in Action: Figure 4 depicts an attacker (i.e., malicious central) attempting to exploit a peripheral, which is vulnerable to *Key Size Overflow* (*CVE-2019-19196*). At start of the BLE connection procedure (*Untrusted Link*), *VaktBLE* intercepts the attacker by selectively jamming the (Link Layer) connection request destined to the peripheral (***step*** 1). Then, *VaktBLE* communicates with the central and queues received packets during the *Link Layer Setup* procedure (***step*** 2). Subsequently, *VaktBLE* starts its exclusive connection with the peripheral (***step*** 3) and forwards the previously queued packets to the peripheral if such packets are valid (***step*** 4). Such an exclusive connection establishes a benevolent MitM bridge used for validating packets destined to the peripheral (Trusted Link).

As illustrated in Figure 4, an attacker proceeds to pairing procedure and attempts to trigger *CVE-2019-19196* by performing two malicious actions: Firstly, it sends a pairing request with an oversized encryption key size k_s (***step*** 5) and secondly, the attacker preemptively starts an out-of-order encryption procedure (***step*** 6). In both cases, *VaktBLE* prevents the attack by blocking the BLE packets associated with the malicious actions, namely ***non-compliant*** encryption key size and ***out-of-order*** encryption start request. The detection of such actions is respectively performed by the *filtering* and *FSM Check* components (see Section 3 for details). Finally, the connection with the malicious central is dropped and *VaktBLE* alerts the user of the attack attempt.

The illustrated attack and defense scenario (Figure 4), across *SMP Pairing* and *Link Layer Encryption* procedures, concretely demonstrates the capability of *VaktBLE* to protect

arbitrary peripherals from attacks in multiple BLE protocols down to the data link (i.e., the *Link Layer*). Consequently, *VaktBLE* is the first practical approach to enable real-time defense against *difficult-to-patch* BLE vulnerabilities such as *Sweyntooth* [2], *BLEDiff* [3] and *CyRC* [4].

3. Design of *VaktBLE*

In this section, we delve into discussing the different key components of *VaktBLE*.

3.1. Hijacking the BLE Connection

We employ a novel and efficient BLE jamming approach to hijack the connection between a central and peripheral. This is done by introducing a non-compliant peripheral that can selectively jam the CRC of BLE *CONNECTION_INDICATION* packet while reading the unmodified *connection parameters* of such packet. We argue that our approach is much more efficient in hijacking BLE connections as compared to other works [8], [7], [13], [14], [9] that rely on jamming and spoofing new advertisements. Contrary to these works, *VaktBLE* only jams a single packet (*connection request*) and does not introduce any redundant packet during the hijacking process.

As shown in Figure 5, jamming only the CRC of *CONNECTION_INDICATION* induces the real peripheral to ignore central connection attempts while keeping packet contents intact. Our non-compliant peripheral ignores the jammed CRC and reads unmodified packet bytes, using correct *CONNECTION_INDICATION* parameters to hijack the central connection. This allows us to silently "steal" the connection while preventing the legitimate peripheral from ever receiving the correct connection request.

Our approach works because the BLE standard does not enforce any mutual validation at the start of the BLE connection procedure. As shown in Figure 5, the central assumes that the *CONNECTION_INDICATION* is correctly received by the legitimate peripheral and establishes a link layer connection upon receiving an arbitrary response (i.e., *anchor point*) from the peripheral within the time window $\left[\Delta_w,\ \Delta_w+\Delta_s\right]$. We note that the anchor point is transmitted by the peripheral in a precise time window and with specific BLE channel parameters such as *CRCInit* and *Hop Interval*. Extracting all parameters from the jammed packet is crucial for the non-compliant peripheral to correctly transmit the spoofed anchor point, ensuring the central continues the data connection.

The selective jamming of the CRC presents two challenges: Firstly, it introduces the risk of corrupting the bits immediately before the start of the three CRC bytes. Secondly, the time it takes for the BLE hardware to switch from reception to transmission (i.e., jamming) is often longer than the time required to finish the over-the-air transmission of the CRC, which is $24us$ at $1mbps$. In summary, if the packet is jammed too early, we risk corrupting relevant bits so that spoofing the anchor point response is impossible. Otherwise, if jamming occurs too late, the CRC will not be corrupted and the legitimate peripheral will transmit the anchor point, thus preventing the hijacking. To address these challenges, the non-compliant peripheral uses the bit-counter feature of the *Nordic nRF52840 System-on-Chip* (SoC) to specify the exact received over-the-air bit in which to trigger a software interrupt. Subsequently, this software interrupt triggers the jamming of the CRC as quickly as possible. This is achieved by using an undocumented feature of the *nRF52* radio hardware, which reduces the time to switch from reception to transmission to about $20us$, just in time to corrupt the last CRC byte (see Figure 14 in the Appendix for details of this implementation). In contrast, switching the radio mode from RX to TX for jamming takes more than $40us$ due to default mode switching sequence (RX ramp down and TX ramp up). We reduce the time for the nRF52 hardware to switch radio modes by directly enabling the radio TX mode (ramp up), without disabling RX mode first (ramp down) (see Figure 5).

3.2. Validating packets down to the Link Layer

Detecting Invalid Packet: *VaktBLE* classifies a BLE packet as *invalid* under two conditions. Firstly, if the bytes within the packet cannot be decoded, then such packet is classified ***malformed***. This occurs with truncated packets or mismatched byte values in the expected protocol structure (i.e., grammar). Secondly, a packet that is fully decoded (i.e., not malformed), but contains a *field value* violating the protocol specification, is categorized as ***non-compliant***. As shown in Figure 8, *VaktBLE* detects *malformed* packets upon receiving an error in the ***Decoding*** component. In contrast, *non-compliant* packets are detected by the ***Filtering*** component.

While *malformed* packets can be indicated by the decoding library (i.e., Bluetooth Wireshark Decoders) as parsing errors, detecting non-compliance in packets is more involved. In such cases, we provide to the ***Filtering*** component, a set of *filtering rules* that contains invariant θ_i for each protocol layer P_L^i of the decoded packet P_D. These invariants specify compliance tests against fields of BLE protocols such as *Link Layer*, *L2CAP*, *SMP* and *ATT*. This invariants ensure that the decoded field values are *semantically* correct according to the Bluetooth Core Specifications [6]. Some invariants θ_i for multiple BLE protocols are presented in Equations 1-4.

$$\theta_i(P_D) = \begin{cases} k_s \geq 7 \wedge k_s \leq 16 & \text{if } P_L^i = \text{SMP} \wedge \text{opcode} = 0x1 \quad (1) \\ 0x01 \leq \text{opcode} \leq 0x1e \ \ldots & \text{if } P_L^i = \text{ATT} \quad (2) \\ \text{len}(P_B) = P_B[23:24] + 4 & \text{if } P_L^i = \text{L2CAP} \quad (3) \\ \Delta_{hop} \geq 5 \wedge \Delta_W \geq 0 \ \ldots & \text{if } P_L^i = \text{LL} \wedge \text{opcode} = 0x5 \quad (4) \\ \ldots & \ldots \quad (5) \end{cases}$$

In Equations 1-4, k_s captures the key size field value, len captures the packet length in bytes and Δ_{hop}, Δ_W are other field values from P_D. In *VaktBLE*, we created a total of only six (6) filtering rules for checking the semantic correctness. After the ***Decoding*** component outputs the decoded packet P_D, the ***Filtering*** component generates invariant of the

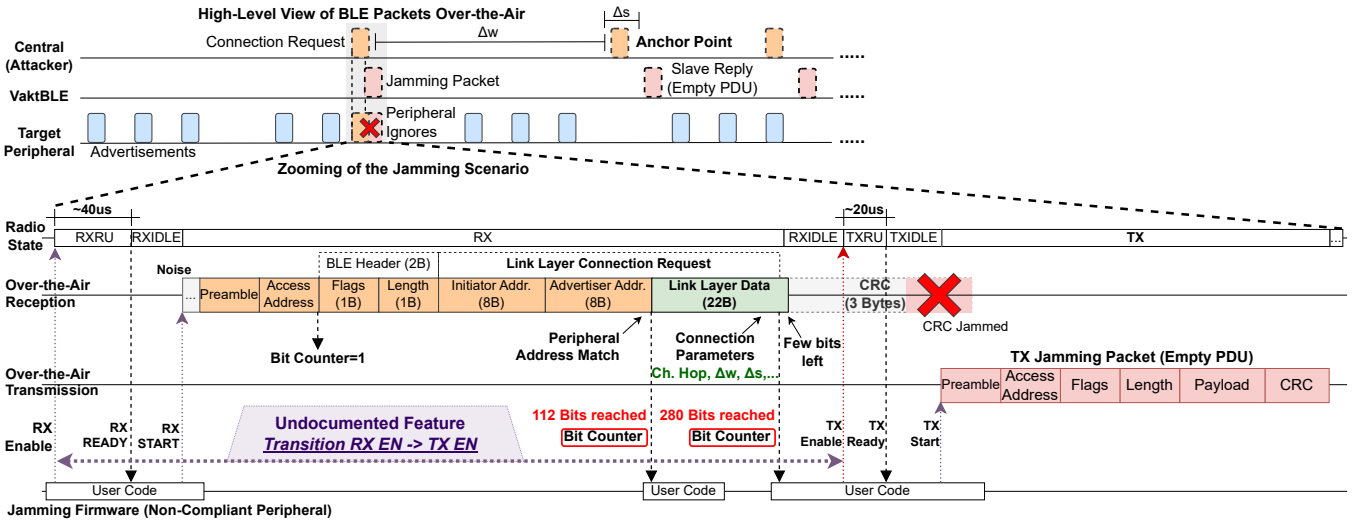

Figure 5. An Illustration of our Novel Selective Jamming Technique. The Non-Compliant Peripheral (i.e., Jamming Firmware) jams the CRC of the connection request, while extracting Connections Parameters such as *Channel Interval*, *Window Offset* (Δ_w), *Window Size* (Δ_s), among others.

decoded packet $\theta_i(P_D)$ that matches with each packet layer P_L^i (rightmost side of Equations 1-4). For example, consider a decoded *SMP Pairing Request* (P'_D) yields the information in line with Equations 6-8. Therefore, such P'_D matches Equations (1) and (3) and results in the invariant $\theta(P'_D)$ of Equation 9. Consequently, $\theta(P'_D)$ is evaluated as *valid* by the *filtering* component and hence the *Pairing Request* is forwarded to the peripheral. In contrast, let us consider the decoded *SMP Pairing Request* (P'_D) resulted the information in line with Equations 6-7, but the key size $k_s = 50$, hence, non-compliant. In this case, $\theta(P'_D)$ evaluates to the logical formula *false* for layer $P_L^i = SMP$, leading to an invalidation of P'_D and termination of the connection with the central.

$$Pairing\ Request \Rightarrow P'_D = \begin{cases} P'_L = [LL, L2CAP, SMP] & (6) \\ opcode = 0x1 & (7) \\ k_s = 16 & (8) \end{cases}$$

$$\theta(P'_D) = (k_s \geq 7 \wedge k_s \leq 16) \wedge (len(P'_B) = P'_B[23:24] + 4) \quad (9)$$

We note that adding more invariants can easily be accomplished by appending new rules, thus making *VaktBLE* flexible to handle more protocols or specific user-defined validation cases.

Detecting Out-of-Order Packet: This detection is crucial to prevent attacks exploiting incorrect handling of protocol message procedures. To this end, *VaktBLE* leverages a state machine model of the pairing procedures to check if a packet addressed to the target is received in the correct sequence (i.e., *FSM Check* component of Figure 7). This ensures, for example, that a malicious central fails to trick a vulnerable peripheral into starting the encryption procedure during an ongoing pairing procedure. Particularly, such approach can effectively stop attacks exploiting known security bypass vulnerabilities such as *Zero LTK Installation*, *DHCheck Skip* [2], *Legacy Pairing Bypass* [3] etc. However, certain Control PDU packets (e.g., *LL_FEATURE_REQ*, *LL_VERSION_IND*, etc.) can be transmitted in different sequences during normal communication. This variability also often depends on the specific implementation by different vendors. Therefore, to avoid yielding false positives due to this variability, *VaktBLE* is specifically designed to ensure that normal communication protocols remain uninterrupted. To this end, *VaktBLE* enforces out-of-order detection *only for sequential procedures* such as pairing and encryption, thus not hindering standard communication process.

Each state machine model instance is associated with a pairing configuration, e.g., legacy pairing and secure connections. Therefore, *VaktBLE* dynamically selects the correct state machine model based on certain pairing messages exchanged between the central and target. Concretely, *VaktBLE* generates state machines for pairing configurations and tracks communication states during pairing. This is achieved by constructing simple *Mapping Rules* that map BLE packet types to protocol states.

Figure 6 provides an example on how our state mapping works. The *mapping rules* illustrate the protocol layer (via `layer.name`) *SMP* and the name of the field (via `field.name`) where packet type is located (i.e., `btsmp.opcode`). When a packet P_1 is analyzed from the offline captures (P_1 is shown above the mapping rules in Figure 6), it's last layer (i.e., *SMP* for P_1) is first used to identify the mapping rule. Since P_1 protocol layer *SMP* matches with the `layer.name` of the mapping rule, the opcode value is extracted from the packet and the corresponding opcode string is obtained from lookup dictionaries (see Figure 6). The state name is then formed with packet direction (i.e., TX), layer name (i.e., *SMP*) and the opcode string (i.e., `Pairing Req.`).

Using the state mapper explained in the preceding paragraph, the workflow for detecting out-of-order packets is illustrated in Figure 7. Firstly, offline captures of different pairing modes, as well as the *Mapping Rules* file, are

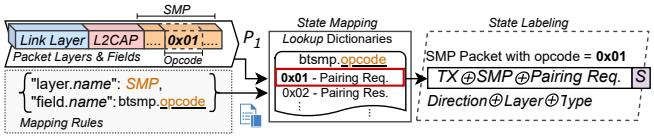

Figure 6. Workflow of state machine generation

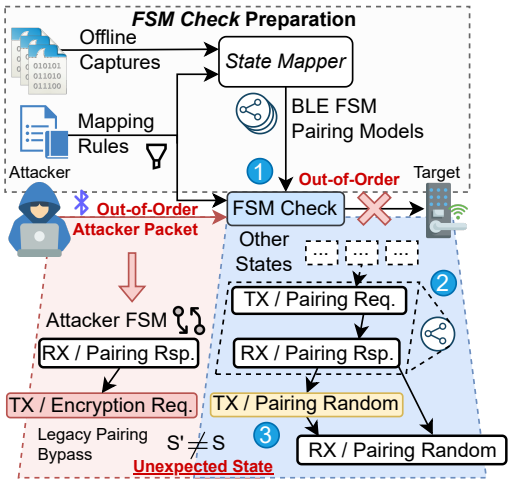

Figure 7. Workflow of *FSM Check* component

provided to the *State Mapper* (step ❶). Then, *FSM Check* selects at runtime the pairing model based on the pair of messages ⟨*Pairing Request*, *Pairing Response*⟩ (step ❷). After the model selection, *FSM Check* validates every response from the central until the pairing and encryption procedure ends in the expected sequence. Otherwise, any packet received at an unexpected state S' is blocked, as shown in step ❸. The aforementioned sequence of events highlight the practicality of *FSM Check* to protect the target against out-of-order attacks exploiting security bypass vulnerabilities e.g., *Legacy Pairing Bypass* [3].

Detecting Flooding: Detecting DoS attacks such as *ATT Sequence Deadlock* [2] and *CyRC Repeated LL Packets* [4] requires tracking the rate of requests that are received in the MitM bridge and subsequently blocking new requests if such rate is higher than expected. Since BLE technology relies on channel hopping, the rate of requests is captured relatively via a counter named *connection event* (CE). This indicates the number of channel hops since the start of the BLE connection (step ❷ of Figure 4). In such case, the *Filtering* component indicates a *flooding attempt* if a request packet is received before a *minimal number of connection events* (CE_{min}) since the reception of previous request. Therefore, the central must satisfy $CE \geq CE_{min}$ throughout the communication with the target. This enforces a fixed *connection event gap* (CE_{gap}) that the central is ought to wait before sending new requests.

Given that each protocol layer allows an ongoing request procedure, CE_{min} is independently instantiated for the last protocol layer of a request. This is so that requests of different protocols are not incorrectly flagged as flooding. For each protocol layer, CE_{min} is initialized to zero. Whenever a request is sent from the (attacker) BLE central, CE_{min} is

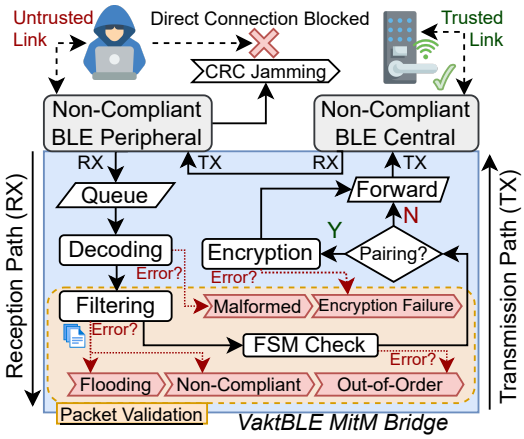

Figure 8. Software Components of *VaktBLE*.

set to CE_{gap}. Thus, by enforcing $CE \geq CE_{min}$, *VaktBLE* ensures that no two consecutive requests within the same protocol layer are sent within CE_{gap} connection events. Violation of the condition $CE \geq CE_{min}$ results in a dropped connection.

Detecting Encryption Failure: Detecting Failures in BLE encryption is required to prevent exploits that purposely fail the encryption integrity check (i.e., *HCI Desync Deadlock* [2]). Such failure can be easily detected by getting the status of the *AES CCM* message integrity check in the *Encryption* component (see Figure 8). Once such check fails in any instance of the BLE communication, no further packets are forwarded to the target, and the central is immediately disconnected from the *VaktBLE* MitM bridge. The component monitors the pairing procedure, deriving the encryption key for communication between peripheral and central devices. This is vital for *VaktBLE* to validate packets in encrypted communication links. While simple secure pairing (SSP) automatically obtains encryption parameters, methods like Passkey Entry require user input of the peripheral's passkey into *VaktBLE*.

3.3. Extensibility of *VaktBLE*

VaktBLE provides the user fine-grained control over BLE protocols to non-invasively enforce security properties of the target peripheral. Our implementation, centered around rule-based and stateful mechanisms, is inherently designed for easy extensibility. Extending or modifying the rules requires minimal knowledge of the BLE protocol and Wireshark and these rules can also be easily maintained by open-source communities as opposed to waiting for proprietary vendor patches. For instance, the user can avoid attacks exploiting *Knob BLE variant* by modifying the invariant of Equation 1 to $k_s = 16$, thus rejecting any reduction to the encryption entropy. Furthermore, the user can add more invariants to the *filtering rules* (see Equations 1-4) and thus force the pairing procedure to only accept a specific configuration such as Secure Connections with *Passkey Entry*. This translates to the invariant $auth.SC = 0x01 \wedge IO_{cap} = 0x03$, where $auth.SC$ constraint ensures that BLE secure connection is

used during pairing and IO_{cap} constraint enforces use of *Passkey Entry* pairing. Subsequently, the user needs to input the known passkey into *Pairing* component before starting *VaktBLE*. This approach is particularly useful against *Key-Size Confusion Attack* and *Method Confusion Attack*, which relies on either changing the encryption key size or mixing different pairing modes [5].

Unilateral Packet Validation: The current implementation of *VaktBLE* is to validate the packets forwarded in a one-way connection stream. We note that our framework does not provide defense against attacks targeting the BLE central [15], although it can be addressed by constructing invariants that account for the central role. Additionally, protecting the central could be achieved by extending *VaktBLE* to block *invalid BLE advertisements* [16] and *scan response* originated from a malicious peripheral. Furthermore, *VaktBLE* has the potential to prevent impersonation of peripherals by jamming all advertisements matching a specific *bdaddress*. While these validations are not discussed in our evaluation, design of spoofing countermeasures is discussed in Section 3.4.

Incompleteness of rules: The validation component of *VaktBLE* employs filtering rules and reference state machines to detect inconsistencies on the Link Layer (LL), Logical Link Control Adaptation Protocol (L2CAP), Attribute Protocol (ATT) and Security Manager Protocol (SMP). In contrast, other parts of the BLE stack, such as advertisement and scan request, are examples of packets not considered in the current implementation. In addition, *VaktBLE* validation rules are projected on the BLE protocol version 5.2. Therefore, future protocol versions may require updating rules, but our open platform allows extending research to enhance the validation component for unconsidered attacks.

3.4. Attacks against *VaktBLE*

In the following, we discuss potential attack vectors of introducing *VaktBLE* in the BLE environment and their countermeasures:

Fake Peripheral: *VaktBLE* leverages a novel selective jamming to establish a bridge. However, jamming techniques that spoof fake peripheral advertisements while blocking the legitimate peripheral could trick *VaktBLE* to establish a bridge to such fake peripheral, hence opening a gap for the attacker to connect to the legitimate peripheral and launch attacks. Such an attack vector is shown in the *leftmost* side of Figure 9 (a).

To defend against the aforementioned attack and its variations, three countermeasures are proposed. Firstly, (i) During startup, *VaktBLE* keeps alive a bridge to the legitimate peripheral all the time, thus preventing the peripheral to accept new connections and hence forcing any central to *always* connect to the *VaktBLE* bridge. This would notably not require hijacking of the connection request, albeit draining more battery of the peripheral due to the continuous connection with *VaktBLE*. (ii) Alternatively, the

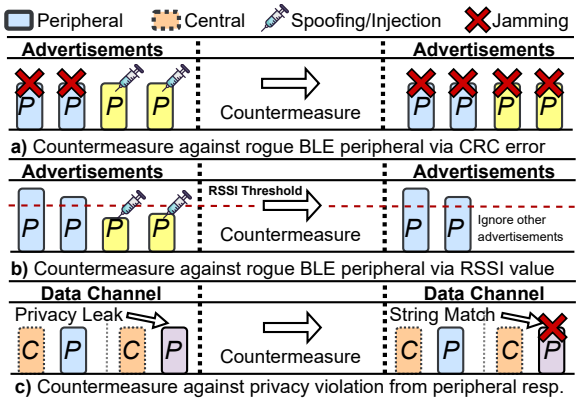

Figure 9. Illustration of attack vectors against *VaktBLE* and their corresponding countermeasure.

non-compliant firmware of *VaktBLE* can inform whether advertisements of the legitimate peripheral are received with CRC errors. Consequently, receiving too many CRC errors can indicate that an attacker is jamming legitimate advertisements and spoofing new advertisements. In such case, the non-compliant peripheral firmware can trigger jamming of any advertisement matching the same address of the legitimate peripheral (see *rightmost* side of Figure 9 (a)). This action would block any connection to the peripheral (legitimate or not). While this countermeasure is not ideal due to self-imposed denial of service, jamming all matching advertisements can temporarily block any exploitation of a vulnerable peripheral while the attacker is nearby and additionally alert the user of such occurrence.

Furthermore, measuring the Received Signal Strength Indicator (RSSI) of advertisements can reveal attack vector which relies on exploiting a race condition of the connection requests when multiple peripherals with the same address are being utilized (*leftmost* side of Figure 9 (b)). In this context, connection requests initiated due to advertisements with an RSSI value lower than a given legitimate peripheral RSSI value threshold, are ignored by *VaktBLE*. As shown in *rightmost* side of Figure 9 (b), this prevents the bridge to connect to rogue peripherals which are normally not close to the peripheral.

Privacy Violation: Finally, *VaktBLE* is susceptible to privacy violation attacks via passive sniffing [1]. Differently than exploitation of the peripheral via active connections, passive sniffing allows an attacker to capture a privacy-sensitive response (e.g., device name) in the link between bridge central and legitimate peripheral. This is because an attacker can send a valid request (ATT request) and passively sniff the over-the-air response on the peripheral side of the bridge (*leftmost* side of Figure 9 (c)). This cannot be initially blocked by *VaktBLE* as its software is not aware of potential privacy violations on the peripheral side. Moreover, the communication between bridge central and peripheral is inherently vulnerable to eavesdrop of plaintext peripheral responses via passive sniffing. To address such issue, *VaktBLE* can destroy peripheral responses according to hardcoded strings provided by the user. Such hardcoded

strings indicate responses that might violate user privacy (e.g., name, device model, etc). Once *VaktBLE* receives an over-the-air payload containing a hardcoded string, selective jamming starts and destroys the rest of the payload. Thus, eavesdroppers will inherently receive a corrupted response. ***Extensions to VaktBLE selective jamming***: *VaktBLE* implements a set of features to activate the discussed countermeasures. Concretely, to avoid rogue peripherals, the non-compliant peripheral informs to *VaktBLE* validation component, the CRC status and RSSI signal of every BLE packet received over-the-air. Moreover, the *non-compliant central* blocks privacy-sensitive payloads by triggering selective jamming. Such payloads match a peripheral response with any user given string provided before *VaktBLE* startup.

Fake *VaktBLE*: It is conceivable that a malicious *VaktBLE* exists within the range of communication, attempting to hijack the connection simultaneously with a benevolent *VaktBLE*. In such scenarios, owing to the characteristics of a BLE connection, multiple collisions are expected to occur, as both *VaktBLE* will attempt to forward packets to the peripheral (i.e., target). Table 1 illustrates two deployments of *VaktBLE* setups (elaborated in Section 4): a malevolent setup and a benevolent setup. Concurrent operation of both setups may lead to numerous collisions, with none of them successfully hijacking the connection. Thus, even if the idea of *VaktBLE* is used for malicious intent, such attacks are unlikely to succeed due to the inherent nature of BLE connections.

Setup	#Connection Attempts	#Connection Collisions	#Successful Connections	#Malevolent Hijack	#Benevolent Hijack
Malevolent close	2800	2778	7	14	1
Malevolent far	2800	2765	4	29	2

TABLE 1. EVALUATION WITH MULTIPLE *VaktBLE* INSTANCES

4. Experimental Setup

VaktBLE implementation is shown in Figure 10. The first setup (a), represents a fully functional *VaktBLE* software with all validation components enabled (i.e., invalid, flooding and out-of-order packets), albeit not portable (anchored). In contrast, setup (b) showcases a lightweight and portable version of *VaktBLE* software which only block malformed packets. While the former setup is used to evaluate the full capabilities of *VaktBLE*, the latter is used to showcase the deployability and extensibility of *VaktBLE* concept to protect portable peripherals. The focus of our evaluation is on the anchored setup, but we showcase the capability of the portable setup in **RQ2**.

Implementation: In Figure 10, the *ESP32-DevKitC* and *ESP32-Wrover-Kit-VE* are the actual peripherals under protection. These modules support Wi-Fi, Bluetooth, and Bluetooth LE for various applications. *VaktBLE* is generic and therefore, the ESP32 peripherals can be replaced with any other device supporting BLE connection, keeping the *VaktBLE* design unchanged. For instance, some of our tested attacks require legacy pairing, which ESP32 devices do not support. Thus, we tested such attacks using Motorola G (5s) as the target.

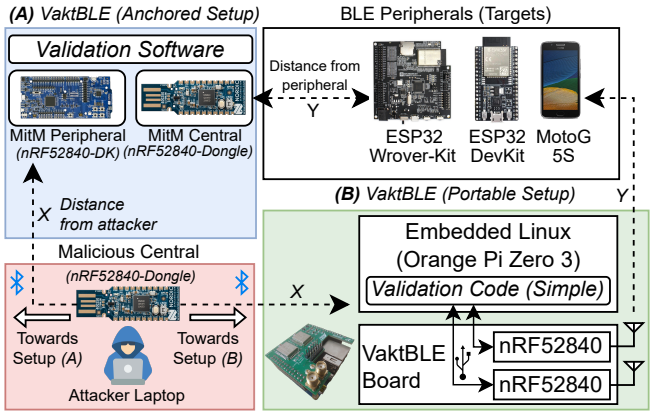

Figure 10. An Illustration of the experimental setup for *VaktBLE*. (a) Anchored setup with full *VaktBLE* software. (b) Portable setup with lightweight *VaktBLE* software.

Default values for "X" and "Y" (Figure 10) are set to 80 cm and 30 cm, respectively. We consider fixed value for "Y" in our experiment as such is the most likely case in real world attacks to the peripheral. We evaluate the sensitivity of *VaktBLE* with respect to distance "X" in **RQ3**. The implementation of *VaktBLE* resides within a peripheral (i.e., Nordic nRF52840-DK), which integrates customized firmware derived from Sweyntooth [2]. This is then paired with a central device (i.e., Nordic nRF52840-Dongle). Specifically, the non-compliant BLE peripheral, which includes the implementation of connection hijacking (see Figure 8), is implemented in Nordic nRF52840-DK. The non-compliant BLE central (see Figure 8), which forwards validated packets to the protected peripherals, is implemented in Nordic nRF52840-Dongle. In summary, the two Nordic devices act as the bridge in the BLE communication between the potential attacker (BLE central) and the target peripheral (i.e., *ESP32* and Motorola G devices in our setup). It is worthwhile to mention that the implementation of our non-compliant BLE peripheral (i.e., Nordic nRF52840-DK) is dedicated to scanning and jamming a single BLE channel. In practice, two more devices are needed to scan all the BLE channels.

The validation software component of the anchored *VaktBLE* setup (see Figure 10(a)) runs on a machine with Intel i7-7700 CPU @ 3.60GHz with 64GB of RAM. The software framework is developed using Python 3.8 (1150 LOC) and C++ (1340 LOC). The Python code contains main script to handle the two threads running non-compliant BLE peripheral and non-compliant BLE central. Additionally, we use Python to implement the functionality for offline processing of .pcap files (to generate the state machine) and for checking validation rules. In contrast, the portable *VaktBLE* setup (Figure 10(b)) is implemented in C++ (457 LOC) and runs on Orange Pi 3 (ARM embedded hardware platform), thus drastically reducing the area of *VaktBLE* setup to about 5x6 cm. This includes the lightweight validation software and two nRF52 radios (non-compliant peripheral and central), which are mounted on top of the embedded platform via

our in-house designed circuit board tailored to *VaktBLE* (green board of Figure 10(b)). Finally, we implement the non-compliant peripheral firmware in C++ across all setups.

Evaluation Setup: We evaluate *VaktBLE* against attacks generated by Sweyntooth [2], BLEDiff [3], CyRC [4] and InjetctaBLE [1]. These attacks target a wide range of BLE protocol layers, such as Secure Manager Protocol (SMP), Link Layer (LL), Logical Link Control Adaptation Protocol (L2CAP) and Attribute protocol (ATT). For the ESP32 targets, we compiled a *nimble* sample code from Espressif v5.0.3 [17], which allowed us to program the ESP32 devices as BLE peripherals. Concurrently, the Motorola G device runs Android 8.1 (OPP28.65-37) version vulnerable to BLEDiff attacks.

5. Results

We answer the following research questions (RQs) to evaluate the effectiveness and efficiency of our *VaktBLE*:

RQ1: How effective is *VaktBLE* in blocking attacks? Table 2 demonstrates the effectiveness of *VaktBLE* in detecting and blocking various attacks generated from state-of-the-art offensive techniques. Attacks on upper protocol layers such as Secure Manager Protocol (SMP) can be blocked, as the encryption process adheres to a well-defined sequence of messages in the standard. Consequently,if a vulnerability exploits the encryption process (e.g., ESP32 HCI Dsync), the encryption validation module in *VaktBLE* detects such exploitation. This prevents a message integrity check (MIC) failure that would cause a denial of service (DoS) attack on the peripheral. Our state machine implementation (see "FSM Check" in Figure 8) detects an out-of-order or an unexpected packet. We evaluated *VaktBLE* with attacks that bypass some steps during the pairing process, such as DHCheck Skip, and several BLEDiff attacks (bypassing legacy pairing and bypassing passkey entry). This demonstrates that *VaktBLE* mitigates attacks on different pairing modes. Most attacks in Table 2 involve invalid packets, detectable by *VaktBLE* through decoding errors or filters (Equations 1-4). *VaktBLE* also identifies malicious retransmissions causing peripheral deadlocks, like Assertion Failure on repeated LL packets (Table 2). These are detected via the flooding detection module (Figure 8). InjectaBLE [1] could have two attack instances in our setup: impersonating the target peripheral, or impersonating the MitM central bridge. Since *VaktBLE* aims to protect the target peripheral, the former is irrelevant. In the latter case, InjectaBLE may send a *LL_CONNECT_UPDATE_REQ* to our MitM central. However, our non-compliant MitM central causes *VaktBLE* to reject this request by design, resulting in a failure of the attack.

We note that *VaktBLE* excludes certain attacks, like those requiring dual BLE and BR/EDR implementation [18], and attacks solely targeting Bluetooth BR/EDR [19].

RQ2: How efficient is *VaktBLE* in real-time detection? We assess *VaktBLE*'s real-time detection capabilities by analyzing overhead (latency) when MitM bridge forwards packets, including Link Layer control packets. We examine delay when forwarding packets from MitM peripheral to MitM central (Figure 10). Timer l_{start} is initiated upon identifying non-malicious packet at MitM peripheral whereas l_{end} indicates the time when the packet is forwarded to MitM central. Thus, overhead $\Delta_t = |l_{end} - l_{start}|$. To compute Δ_t, we launch a total of 250 attack attempts for a total of 25 considered BLE attacks. For this experiment, we used default connection parameters (e.g., *connInterval*, *WinSize*, *WinOffset*) for all tests. This approach enabled us to compute the average overhead introduced by *VaktBLE* as an MitM bridge in the BLE communication process.

The distribution of MitM bridge overhead Δ_t is illustrated in Figure 11, while the overhead for each attack is detailed in Table 2. Table 2 also shows the time to hijack the channel. This is done by setting a reference time t_{start} at the initial anchor point (e.g., *EMPTY_PDU*) sent by the malicious central following the *CONNECTION_INDICATION*. We then determine t_{end} as when the MiTM peripheral in the *VaktBLE* bridge (see Figure 10) sends its first non-empty Link Layer packet. Thus, after the duration $\Delta t_h = t_{\text{end}} - t_{\text{start}}$, we can confirm the end of ***step*** (2), as illustrated in Figure 4. The end of this step, in turn, confirms the completion of the hijack process.

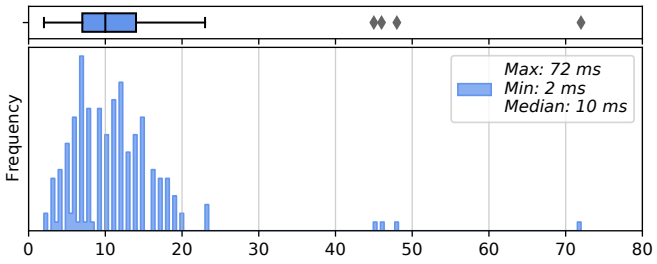

Figure 11. Distribution of *VaktBLE* bridging latency.

Considering the overall performance of *VaktBLE*, including Scapy's latency, we observe a median latency of $10ms$ (Figure 11), which is reasonable given the Bluetooth core specifications [6] allow a minimum connection interval of $7.5ms$ and a maximum of $4000ms$. For attacks exploiting the peripheral by sending invalid connection requests (i.e., $CONNECTION\_INDICATION$), *VaktBLE*'s channel jamming blocks these invalid packets and drops the connection, causing no measurable overhead (N.A in Table 2).

Efficiency of the portable setup: We measured the overhead for the Portable setup of *VaktBLE* (Figure 10(b)). Using the BLE KNOB Variant (CVE-2019-9506), our evaluations showed a reduction in overhead from $10ms$ to $373\mu s$ in the Portable Setup, demonstrating *VaktBLE*'s feasibility for protecting BLE devices in the wild.

RQ3: How robust is *VaktBLE* w.r.t attacker distance? We conducted experiments at various distances ("X" in Figure 10) to represent potential attack scenarios. For this, we use the BLE KNOB Variant with default connection parameters, as in **RQ2**. Figure 12 shows that *VaktBLE* almost always hijacks the connection within the $30cm$-$15m$ range. Even at $2cm$ away (an unlikely situation), *VaktBLE* hijacks

	Attack Name	*VaktBLE* Validation Type				Connectivity	*VaktBLE* Overhead (avg)	Time to hijack the channel (avg)	Target Peripheral
		Malformed/ Non-compliant	Flooding	State Machine Model	Encryption				
Sweyntooth	CVE-2019-16336 - Link Layer Length Overflow	✓				10/10	18.3 ms	40.3 ms	ESP32-Wrover-Kit-VE
	CVE-2019-19195 - Invalid L2cap fragment	✓				10/10	9.2 ms	60.2 ms	ESP32-DevKitC
	CVE-2019-17060 - LLID Deadlock	✓				10/10	15.9 ms	55.5 ms	ESP32-Wrover-Kit-VE
	CVE-2019-17517 - Truncated L2CAP	✓				10/10	7.2 ms	58.5 ms	ESP32-Wrover-Kit-VE
	CVE-2019-17518 - Silent Length Overflow	✓				10/10	12.3 ms	60.2 ms	ESP32-Wrover-Kit-VE
	CVE-2019-17520 - Public Key Crash			✓/Legacy Pairing		10/10	7.1 ms	60.1 ms	ESP32-Wrover-Kit-VE
	CVE-2019-19193 - Invalid Connection Request	✓				10/10	*N.A*	*N.A*	ESP32-Wrover-Kit-VE
	CVE-2019-19192 - Sequential ATT Deadlock		✓			10/10	12.5 ms	60.2 ms	ESP32-Wrover-Kit-VE
	CVE-2019-19196 - Key Size Overflow	✓				10/10	6.8 ms	50.2 ms	ESP32-Wrover-Kit-VE
	CVE-2019-19194 - Zero LTK Installation			✓/Legacy Pairing		10/10	10.5 ms	60.1 ms	MotoG 5S
	CVE-2020-13593 - DHCheck Skip			✓/Secure Connection		10/10	11.3 ms	60.2 ms	MotoG 5S
	CVE-2020-13595 - ESP32 HCI Desync				✓	10/10	7.5 ms	40.8 ms	ESP32-Wrover-Kit-VE
	CVE-2020-10061 - Zephyr Invalid Sequence	✓				10/10	*N.A*	*N.A*	ESP32-Wrover-Kit-VE
	CVE-2020-10069 - Invalid Channel Map	✓				9/10	*N.A*	*N.A*	ESP32-Wrover-Kit-VE
Non-Sweyntooth	InjectaBLE - Hijacking the Peripheral via Central Impersonation (MitM)	-	-	-	-	10/10	*N.A*	*N.A*	ESP32-DevKitC
	CVE-2019-9506 - BLE KNOB Variant	✓				10/10	8.2 ms	60.5 ms	ESP32-Wrover-Kit-VE
	BLEDiff - (E1) Bypassing passkey entry in legacy pairing			✓/Legacy Pairing		9/10	5.4 ms	32.5 ms	MotoG 5S
	BLEDiff - (E3) Bypassing legacy pairing			✓/Legacy Pairing		10/10	7.2 ms	40.6 ms	MotoG 5S
	BLEDiff - (O1) Accepting key size greater than max	✓				10/10	13.2 ms	58.5 ms	ESP32-Wrover-Kit-VE
	BLEDiff - (E4) Accepts *DHKeyCheckSend* with all fields zero	✓				10/10	8.9 ms	60.2 ms	MotoG 5S
	BLEDiff - (E6) Unresponsiveness with *ConReqTimeoutZero* and *ConReqIntervalZero*	✓				10/10	15.5 ms	62.5 ms	MotoG 5S
	CVE-2021-3431 - CyRC Assertion failure on certain repeated LL packets		✓			10/10	10.9 ms	60.3 ms	ESP32-Wrover-Kit-VE
	CVE-2021-3430 - CyRC Assertion failure on repeated LL_CONNECTION_PARAM_REQ		✓			10/10	11.5 ms	60.2 ms	ESP32-Wrover-Kit-VE
	CVE-2021-3433 - CyRC Invalid channel map in CONNECT_IND results to deadlock	✓				8/10	*N.A*	*N.A*	ESP32-Wrover-Kit-VE
	CVE-2021-3434 - CyRC L2CAP: Truncated L2CAP K-frame causes assertion failure	✓				10/10	8.1 ms	46.3 ms	ESP32-DevKitC

TABLE 2. EFFECTIVENESS OF *VaktBLE* IN STOPPING BLE ATTACKS. $\mathbf{CE_{gap} = 4}$ FOR FLOODING VALIDATION. *VaktBLE* EFFICIENCY IS NOT APPLICABLE (NA) WHEN ATTACKS RELY ON $CONNECTION\_INDICATION$, LEADING TO CONNECTION DROP AND NO MEASURABLE OVERHEAD.

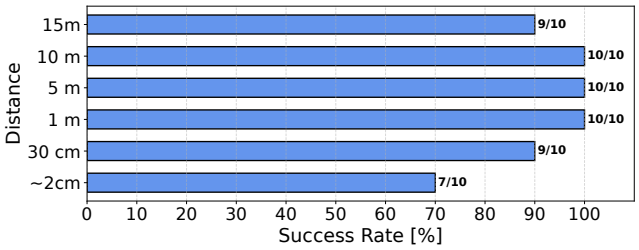

Figure 12. *VaktBLE* connection success rate w.r.t attacker distance from peripheral.

Attack	*VaktBLE* detection	Time to hijack the channel	Target
Link Layer Length Overflow (CVE-2019-16336)	10/10	40.1 ms	ESP32-DevKitC
Invalid L2cap fragment (CVE-2019-19195)	10/10	59.5 ms	ESP32-DevKitC
LLID Deadlock (CVE-2019-17061,CVE-2019-17060)	10/10	55.2 ms	ESP32-DevKitC
Truncated L2CAP (CVE-2019-17517)	10/10	58.9 ms	ESP32-DevKitC
Silent Length Overflow (CVE-2019-17518)	10/10	60.1 ms	ESP32-DevKitC
Public Key Crash (CVE-2019-17520)	10/10	60.2 ms	ESP32-DevKitC
Invalid Connection Request (CVE-2019-19193)	9/10	-	ESP32-DevKitC
Sequential ATT Deadlock (CVE-2019-19192)	10/10	60.1 ms	ESP32-DevKitC
Key Size Overflow (CVE-2019-19196)	10/10	55.2 ms	ESP32-DevKitC
Zero LTK Installation (CVE-2019-19194)	10/10	60.2 ms	ESP32-DevKitC
DHCheck Skip (CVE-2020-13593)	10/10	60.1 ms	Moto G (5s)
ESP32 HCI Desync (CVE-2020-13595)	10/10	41.1 ms	Moto G (5s)
Zephyr Invalid Sequence (CVE-2020-10061)	10/10	-	ESP32-DevKitC
Invalid Channel Map (CVE-2020-10069,CVE-2020-13594)	10/10	-	ESP32-DevKitC

TABLE 3. *VaktBLE* EFFECTIVENESS W.R.T CONTINUOUS ATTACKS.

70% of connection attempts. These results demonstrate *VaktBLE*'s robustness to attacker distance. A range extender [20] could further improve *VaktBLE*'s jamming capability in the future.

RQ4: How *VaktBLE* compares with patching-based tools? We tested state-of-the-art countermeasures involving Bluetooth stack debloating [11]. Since these techniques typically do not protect against link-layer attacks, we chose six representative attacks from Table 2 to validate this (Invalid L2cap fragment, LLID Deadlock, Truncated L2CAP, Sequential ATT Deadlock, Key Size Overflow, and Zero LTK Installation). Using the default connection parameters from **RQ2**, our evaluations showed that debloating [11] fails to safeguard against any of these attacks. Moreover, using such debloater involves understanding of Bluetooth profiles, and much effort is needed to reverse engineer binaries to extract the memory entry point of the Bluetooth handler address for proposed stacks (e.g., Zephyr Project [21]). Additionally, debloaters rely on known function names, meaning state-of-the-art Bluetooth firmware debloaters do not offer out-of-the-box deployment such as *VaktBLE*.

RQ5: How effective is *VaktBLE* w.r.t adaptive attacks? We evaluate *VaktBLE* against the following adaptive attacks:

1) Continuous attacks: In this experiment, we aim to assess *VaktBLE*'s effectiveness against continuous attacks, where successive attacks are launched without waiting for the previous attempt to conclude. During the evaluation of **RQ2**, exploits were launched one at a time. For continuous attacks, we modified Sweyntooth exploits [2] to facilitate their continuous execution ten times each (see Table 3). For instance, if $e_1, e_2, \ldots, e_n$ are the set of exploits, then the script launched a continuous attack in the order: $e_1^{10} \rightarrow e_2^{10} \rightarrow \ldots \rightarrow e_n^{10}$. From Table 3, we observe that *VaktBLE* remains effective even against continuous attacks: it detected all but one attack attempt. Due to *VaktBLE*'s jamming component, attackers cannot initiate new connections while one is ongoing with the target peripheral. Thus, even during continuous attacks, *VaktBLE* drops connections for subsequent attempts in-flight.

2) Mixing benign connections: In this experiment, we introduce a probabilistic behavior in terms of launching malicious (attacks) and benign cases. Benign cases strictly involve connections that follow the BLE specification procedures [6]. We assume that the selection of attacks or benign cases is independent. Thus, we can model it using a binomial distribution. Let X represent the random variable for the number of selected attacks out of 28 total cases (i.e., 14 attacks and 14 benign cases) in n trials. X follows a binomial distribution with parameters n and the probability of selecting an attack ($P(\text{attack})$). The binomial distribution's probability mass function estimates the likelihood of obtaining a certain number of attacks: $P(X = k) = \binom{n}{k} P(\text{attack})^k (1 - P(\text{attack}))^{n-k}$, where $\binom{n}{k}$ is the binomial coefficient, enumerating ways to select k successes from n trials. Here, k is the number of selected attacks in n trials. Table 4 showcases the results of this experiment. Apart from stopping all attack attempts, we also notice that *VaktBLE* does not report any false positives in detection (0/139).

Attack	*VaktBLE* detection	Time to hijack the channel	Target
Link Layer Length Overflow (CVE-2019-16336)	12/12	42.3 ms	ESP32-DevKitC
Invalid L2cap fragment (CVE-2019-19195)	9/9	60.5 ms	ESP32-DevKitC
LLID Deadlock (CVE-2019-17061,CVE-2019-17060)	8/8	58.2 ms	ESP32-DevKitC
Truncated L2CAP (CVE-2019-17517)	11/11	56.9 ms	ESP32-DevKitC
Silent Length Overflow (CVE-2019-17518)	7/7	60.2 ms	ESP32-DevKitC
Public Key Crash (CVE-2019-17520)	13/13	60.2 ms	ESP32-DevKitC
Invalid Connection Request (CVE-2019-19193)	6/6	-	ESP32-DevKitC
Sequential ATT Deadlock (CVE-2019-19192)	10/10	60.1 ms	ESP32-DevKitC
Key Size Overflow (CVE-2019-19196)	5/5	58.2 ms	ESP32-DevKitC
Zero LTK Installation (CVE-2019-19194)	14/14	60.1 ms	ESP32-DevKitC
DHCheck Skip (CVE-2020-13593)	4/4	60.1 ms	Moto G (5s)
ESP32 HCI Desync (CVE-2020-13595)	15/15	42.3 ms	Moto G (5s)
Zephyr Invalid Sequence (CVE-2020-10061)	10/10	-	ESP32-DevKitC
Invalid Channel Map (CVE-2020-10069,CVE-2020-13594)	16/16	-	ESP32-DevKitC
Benign cases	0/139	60.2 ms	ESP32-DevKitC

TABLE 4. *VaktBLE* EFFECTIVENESS W.R.T. MIXED BENIGN/ATTACKS.

3) *VaktBLE* vs automated testing: We validate *VaktBLE* against unknown fuzzing-based BLE attacks using Sweyntooth fuzzer [2] which mutates/duplicates packets sent to peripheral. Fuzzer ran for 1 hour (≈ 1200 iterations). Results are shown in Figure 13. We note that packet mutations may yield invalid packets (malformed or non-compliant), while duplication causes flooding. The blocking of such attacks are embodied in the design of *VaktBLE* (see Figure 8). Figure 13 shows that in an one hour fuzzing session, *VaktBLE* missed only 2.5% (30/1200) of the potential attacks launched by the fuzzer.

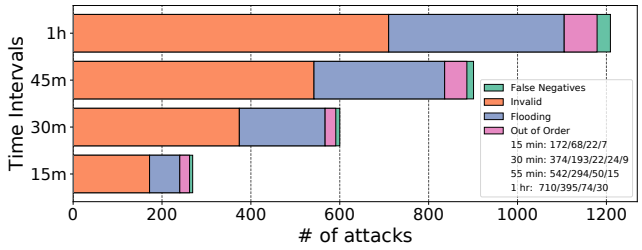

Figure 13. *VaktBLE* effectiveness w.r.t. fuzzing-based attacks. Different categories of detected attacks and the false negatives are shown in the legend. For example, after one hour fuzzing session, *VaktBLE* detected 710 invalid, 395 flooding and 74 out-of-order message attacks, while 30 remaining attacks managed to be successful.

4) Different connection parameters: Finally, the attacker can vary the connection parameters. We validate *VaktBLE*'s stability by varying `CONNECTION_INDICATION` packet parameters (*connInterval, WinSize, Hop, cHM*). We use the *CVE-2019-9506: BLE KNOB* variant like **RQ3**, with connection parameters modified per Table 5 (10 iterations per parameter). We note that for elevated values of the *WindowSize* (i.e., last row in Table 5), the non-compliant central in *VaktBLE* MitM bridge fails to establish a connection with the target peripheral. Thus, even though *VaktBLE* framework becomes unstable in such cases, it still prevents the intended attack.

Connection parameter	Range	Connectivity
Low hop interval	5-10	10/10
High hop interval	11-16	10/10
Low Connection Interval	8.75 - 20 ms	10/10
High Connection Interval	125 - 137.5 ms	10/10
Random Channel Mask	0x0000000000 - 0x07FFFFFFFF	10/10
Low Window Size	0 - 10 ms	10/10
High Window Size	11.25 - 17.25 ms	*

TABLE 5. *VaktBLE* EFFECTIVENESS W.R.T CONNECTION PARAMETERS.

6. Discussion and Future Outlook

In the following discussion, we explore the limitations of our current evaluation.

Choice of Attacks: Since *VaktBLE* primarily validates link-layer packets, we evaluated attacks from tools such as SweynTooth, BLEDiff, CyRC, and InjectaBLE that target link layers and have reproducible proofs-of-concept (PoCs). Additionally, we intended to show *VaktBLE*'s robustness against arbitrarily mutated, flooded, and out-of-order packets beyond the 25 evaluated attacks (summarized in Table 2). This involved evaluating *VaktBLE* against automated testing tools, such as a BLE Fuzzer (better detailed in Section 5). We note that not all fuzzer-generated communications are actual attacks. However, most of the evaluated attacks are originated from such packets.

Unable to enhance the security of existing target: Although *VaktBLE* ensures the integrity of the exchanged packets, this does not necessarily translate to an increased or enhanced level of connection security. In future, we aim to enhance the security of a BLE connection by leveraging the foundation of *VaktBLE*.

Scalability of jamming: While our current jamming implementation is designed for an individual BLE channel and may not scale efficiently to defend thousands of devices simultaneously, our jamming strategy is highly adaptable and extensible. This approach can be employed and integrated into software-defined radios (SDRs), potentially enabling protection for a broader range of devices and scenarios.

Sustained drops of connection: *VaktBLE* packet validation process is configured to jam and terminate the connection as soon as a packet is invalidated. However, this disconnection process could be explored by a malicious device, which may attempt to send a malformed packet many times. Thus, even though such attempt will fail to exploit any vulnerability in the target peripheral, it may cause a battery drain attack on the target or defense bridge. However, we note that *VaktBLE* does not contribute to attackers in degrading *user experience* via sustained connection drops. This attack surface already exists within BLE. Nonetheless, we believe that by blocking direct connection to the peripheral even under repeated attempts from the attacker, VaktBLE prevents more critical vulnerabilities such as remote code execution.

Support to Multiple Peripherals: Implementing BLE connection time slicing could enable support for multiple target peripherals per *VaktBLE* instance. However, this might allow an attacker to directly connect to the target peripheral when the non-compliant peripheral is not listening to advertisement channels. This is because the radio of *nRF52840* only communicates in one channel at a time.

Setup Footprint: Currently, the setup of *VaktBLE* involves four devices per target (three non-compliant peripherals, one for each channel and one non-compliant central). However, implementing techniques to follow advertisements of the target, could reduce *VaktBLE* to one non-compliant peripheral per target. This results in only two devices for *VaktBLE*.

7. Related Work

BLE Attacks and Testing: In recent years, Bluetooth Low Energy (BLE) has witnessed a rise in new attack vectors [22], [1], [4], [2], [3], [5]. *VaktBLE* responds by offering real-time countermeasures against known and potentially unknown BLE vulnerabilities to defend against malicious actors within the radio range. In the past decade, there has also been numerous frameworks to uncover BLE vulnerabilities [3], [2], [23], [24], [25]. While L2Fuzz [23] targets the L2CAP layer, several works have exposed BLE link-layer vulnerabilities [2], [3]. However, in contrast to *VaktBLE*, these techniques primarily concentrate on testing and analysis, they do not propose real-time defense against BLE attacks.

Work	BLE Support	Reverse Engineering	Stateful Validation	Flooding Validation	Structure Validation	Link Layer Support	COTS Support	Attack Model Applicability
LIGHTBLUE [11]	●	◗	●	❍	●	❍	◗	Any Role
BlueShield [10]	●	❍	❍	❍	●	❍	●	Only Central
Battery Exhaustion Prevention [26]	◗	❍	❍	❍	❍	❍	●	Any Role
BLE-guardian [27]	●	❍	❍	❍	●	❍	●	Only Peripheral
Inside Job [28]	●	●	❍	❍	●	❍	●	Only Peripheral
MagicPairing [29]	◗	●	❍	❍	❍	❍	❍	Only Peripheral
FirmXRay [30]	◗	●	❍	❍	●	●	●	Any Role
OASIS [31]	●	●	❍	❍	●	●	◗	Any Role
VaktBLE	●	❍	●	●	●	●	●	Only Peripheral

TABLE 6. *VaktBLE* VS OTHER DEFENSE. ●: FULL CONSIDERATION, ❍: EXCLUSION, ◗: PARTIAL CONSIDERATION. FOR THE *COTS Support* COLUMN ◗ CAPTURES COMPATIBILITY WITH ARBITRARY COTS DEVICES.

BLE Defense: Table 6 positions *VaktBLE* with respect to existing defense strategies. Notably, while BlueShield [10] presents defense mechanisms against spoofing attacks, it is not applicable to attacks that exploit Link Layer (LL) vulnerabilities within the peripheral [2], [3]. This is because BlueShield considers an attack model (see last column of Table 6) where the attacker, acting as the peripheral, broadcasts advertisement packets to spoof the victim (central). While BlueShield is capable of detecting such attacks, it cannot prevent them in real-time. *VaktBLE* takes a different approach by protecting peripheral devices against attacks from a malicious BLE central. In this scenario, the attacker (malicious central) does not broadcast advertisement packets before launching an attack. As a result, BlueShield's approach is unable to detect these kinds of attacks because the malicious central does not exhibit detectable behavior through advertisement broadcasting.

Previous works such as LightBlue [11] has targeted patching the Bluetooth stack by creating a hook in the device's firmware. However, this approach patches vulnerabilities above link layer and requires access to the device's firmware binary. In contrast, *VaktBLE* includes protection against link layer attacks, and it is applicable for arbitrary COTS targets, thus not requiring expertise from the user. It is worthwhile to mention that the work preventing battery-exhaustion attacks [26] relies only on BLE-Mesh networks, thus, it is categorized with partial BLE support (◗).

Furthermore, approaches using embedded defensive mechanisms [31], require modifying both the device firmware and controller instrumentation. The latter, involving changes to the controller's software, is particularly challenging and demands significant reverse engineering effort. Consequently, approaches leveraging reverse engineering techniques [28], [29], [30] become less practical for protecting already deployed and non-patched devices.

Jamming: In contrast to *VaktBLE*, state-of-the-art jamming fails to perform selective jamming in a way to maintain the contents of the jammed payload [13], [14], [7]. This is essential to start the MitM bridge. Other approaches [32] can only hijack established communications, which is too late to defend Link-layer attacks. Additionally, *VaktBLE* targets a single message with an efficient approach that does not spoof the advertisement channels. Therefore, BLE Monitoring approaches based on analyzing advertisements [8], [27] are not capable to detect our MitM bridge at all. We note that, due to the timing overhead, existing jamming techniques [13], [14], [7] have to start the jamming process early, resulting in a situation where several bytes of the payload cannot be preserved. *VaktBLE* addresses such challenges by proposing an efficient jamming technique that finishes in about the transmission time required for three bytes (i.e., length of CRC). Additionally, work attempting to jam existing connections, involving parameter guessing [7], may often fail to hijack connections due to specific race conditions. In contrast, *VaktBLE* reliably and deterministically hijacks a new connection by abusing the start of it, due to an oversight in the BLE standard.

In summary, previous techniques were able to perform jamming either before BLE connections (e.g., advertisement jamming [8]), or after connections (e.g., InjetaBLE [1]). In contrast, *VaktBLE* is the first to selectively jam the start of a connection, providing a predictable result for hijacking.

8. Conclusion

This paper presents *VaktBLE*, a novel and practical selective jamming technique and software framework to comprehensively defend arbitrary BLE peripherals in real-time and in a over-the-air fashion. This, finally allows users the option to protect the *link layer* of devices that can take arbitrarily long to receive updates or might never get fixed at all. *VaktBLE* is the first defense of black-box BLE peripherals that can protect the link layer in combination with any BLE protocols. We hope that our proposed real-time *"benevolent"* MitM bridge can also be employed as a fundamental platform in other works for real-time BLE intrusion detection and mitigation, security enhancements to link layer and blacklisting of untrustworthy BLE centrals, among others. For reproduction and advancing the research in this area, *VaktBLE* is publicly available in the following URL: https://github.com/asset-group/vakt-ble-defender

Acknowledgement: This research is partially supported by MOE Tier 2 grant (Award number MOE-T2EP20122-0015) and National Research Foundation, Singapore, under its National Satellite of Excellence Programme "Design Science and Technology for Secure Critical Infrastructure: Phase II" (Award No: NRF-NCR25-NSOE05-0001). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of the respective funding agencies.

References

[1] 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.

[2] 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, July 2020, pp. 911–925. [Online]. Available: https://www.usenix.org/conference/atc20/presentation/garbelini

[3] I. Karim, A. Ishtiaq, S. Hussain, and E. Bertino, "Blediff: Scalable and property-agnostic noncompliance checking for ble implementations," in *2023 IEEE Symposium on Security and Privacy (SP)*. Los Alamitos, CA, USA: IEEE Computer Society, may 2023, pp. 3209–3227. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SP46215.2023.10179330

[4] M. Karhumaa, "Cyrc vulnerability advisory: Denial-of-service vulnerabilities in zephyr bluetooth le stack," https://www.synopsys.com/blogs/software-security/cyrc-advisory-zephyr-vulnerability/, 2021.

[5] M. Shi, J. Chen, K. He, H. Zhao, M. Jia, and R. Du, "Formal analysis and patching of BLE-SC pairing," in *32nd USENIX Security Symposium (USENIX Security 23)*. Anaheim, CA: USENIX Association, Aug. 2023, pp. 37–52. [Online]. Available: https://www.usenix.org/conference/usenixsecurity23/presentation/shi-min

[6] B. SIG, "Bluetooth core specification v5.2," https://www.bluetooth.com/specifications/specs/core-specification-5-2/, Dec. 2019.

[7] 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. [Online]. Available: https://hal.laas.fr/hal-02346074

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

[9] T. Bamelis, "Jambler: selectively jamming ble5 csa# 2 by deducing connection parameters." Master's thesis, KU Leuven. Faculteit Ingenieurswetenschappen, 2021.

[10] 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)*. San Sebastian: USENIX Association, Oct. 2020, pp. 397–411. [Online]. Available: https://www.usenix.org/conference/raid2020/presentation/wu

[11] J. Wu, R. Wu, D. Antonioli, M. Payer, N. O. Tippenhauer, D. Xu, D. J. Tian, and A. Bianchi, "LIGHTBLUE: Automatic profile-aware debloating of Bluetooth stacks," in *Proceedings of the USENIX Security Symposium (USENIX Security)*, Aug. 2021.

[12] Mitre, "Mitre CVE Database," https://cve.mitre.org/, 2024.

[13] M. Ryan, "Bluetooth: With low energy comes low security," in *7th USENIX Workshop on Offensive Technologies (WOOT 13)*. Washington, D.C.: USENIX Association, Aug. 2013. [Online]. Available: https://www.usenix.org/conference/woot13/workshop-program/presentation/ryan

[14] M. von Tschirschnitz, L. Peuckert, F. Franzen, and J. Grossklags, "Method confusion attack on bluetooth pairing," in *2021 IEEE Symposium on Security and Privacy (SP)*, 2021, pp. 1332–1347.

[15] 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. [Online]. Available: https://www.usenix.org/conference/woot20/presentation/wu

[16] A. Gangwal, S. Singh, R. Spolaor, and A. Srivastava, "Blewhisperer: Exploiting ble advertisements for data exfiltration," in *Computer Security – ESORICS 2022: 27th European Symposium on Research in Computer Security, Copenhagen, Denmark, September 26–30, 2022, Proceedings, Part I*. Berlin, Heidelberg: Springer-Verlag, 2022, p. 698–717. [Online]. Available: https://doi.org/10.1007/978-3-031-17140-6_34

[17] Espressif, "Espressif IoT development framework," https://github.com/espressif/esp-idf/tree/v5.0.1, February 2023, commit ID: a4afa44.

[18] D. Antonioli, N. O. Tippenhauer, K. Rasmussen, and M. 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*, ser. ASIA CCS '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 196–207. [Online]. Available: https://doi.org/10.1145/3488932.3523258

[19] M. Ai, K. Xue, B. Luo, L. Chen, N. Yu, Q. Sun, and F. Wu, "Blacktooth: Breaking through the defense of bluetooth in silence," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 55–68. [Online]. Available: https://doi.org/10.1145/3548606.3560668

[20] N. Semiconductor, "Nordic range extenders," https://www.nordicsemi.com/products/range-extenders/, 2023.

[21] Zephyr, "Zephyr Bluetooth Stack Architecture," https://docs.zephyrproject.org/latest/connectivity/bluetooth/bluetooth-arch.html, 2024.

[22] J. Wu, R. Wu, D. Xu, D. Tian, and A. Bianchi, "Sok: The long journey of exploiting and defending the legacy of king harald bluetooth," in *2024 IEEE Symposium on Security and Privacy (SP)*. Los Alamitos, CA, USA: IEEE Computer Society, may 2024, pp. 23–23. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SP54263.2024.00023

[23] H. Park, C. K. Nkuba, S. Woo, and H. Lee, "L2fuzz: Discovering bluetooth l2cap vulnerabilities using stateful fuzz testing," in *2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, jun 2022. [Online]. Available: https://doi.org/10.1109%2Fdsn53405.2022.00043

[24] A. Pferscher and B. K. Aichernig, "Stateful black-box fuzzing of bluetooth devices using automata learning," in *NASA Formal Methods Symposium*. Springer, 2022, pp. 373–392.

[25] A. Ray, V. Raj, M. Oriol, A. Monot, and S. Obermeier, "Bluetooth low energy devices security testing framework," in *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*, 2018, pp. 384–393.

[26] Z. Guo, I. G. Harris, Y. Jiang, and L.-f. Tsaur, "An efficient approach to prevent battery exhaustion attack on ble-based mesh networks," in *2017 International Conference on Computing, Networking and Communications (ICNC)*, 2017, pp. 1–5.

[27] K. Fawaz, K.-H. Kim, and K. G. Shin, "Protecting privacy of BLE device users," in *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, aug 2016, pp. 1205–1221. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/fawaz

[28] J. Classen and M. Hollick, "Inside job: Diagnosing bluetooth lower layers using off-the-shelf devices," in *Proceedings of the 12th Conference on Security and Privacy in Wireless and Mobile Networks*, ser. WiSec '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 186–191. [Online]. Available: https://doi.org/10.1145/3317549.3319727

[29] D. Heinze, J. Classen, and F. Rohrbach, "Magicpairing: Apple's take on securing bluetooth peripherals," in *Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, ser. WiSec '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 111–121. [Online]. Available: https://doi.org/10.1145/3395351.3399343

[30] H. Wen, Z. Lin, and Y. Zhang, "Firmxray: Detecting bluetooth link layer vulnerabilities from bare-metal firmware," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 167–180. [Online]. Available: https://doi.org/10.1145/3372297.3423344

[31] R. Cayre, V. Nicomette, G. Auriol, M. Kaâniche, and A. Francillon, "OASIS: An Intrusion Detection System Embedded in Bluetooth Low Energy Controllers," in *2024 ACM Asia conference on Computer and Communications Security (ASIACCS).*, Singapore, Singapore, July 2024. [Online]. Available: https://hal.science/hal-04488826

[32] D. Cauquil, "Btlejack: a new bluetooth low energy swiss-army knife," https://github.com/virtualabs/btlejack/tree/v2.1.1, November 2022, commit ID: d0dd2df.

Appendix

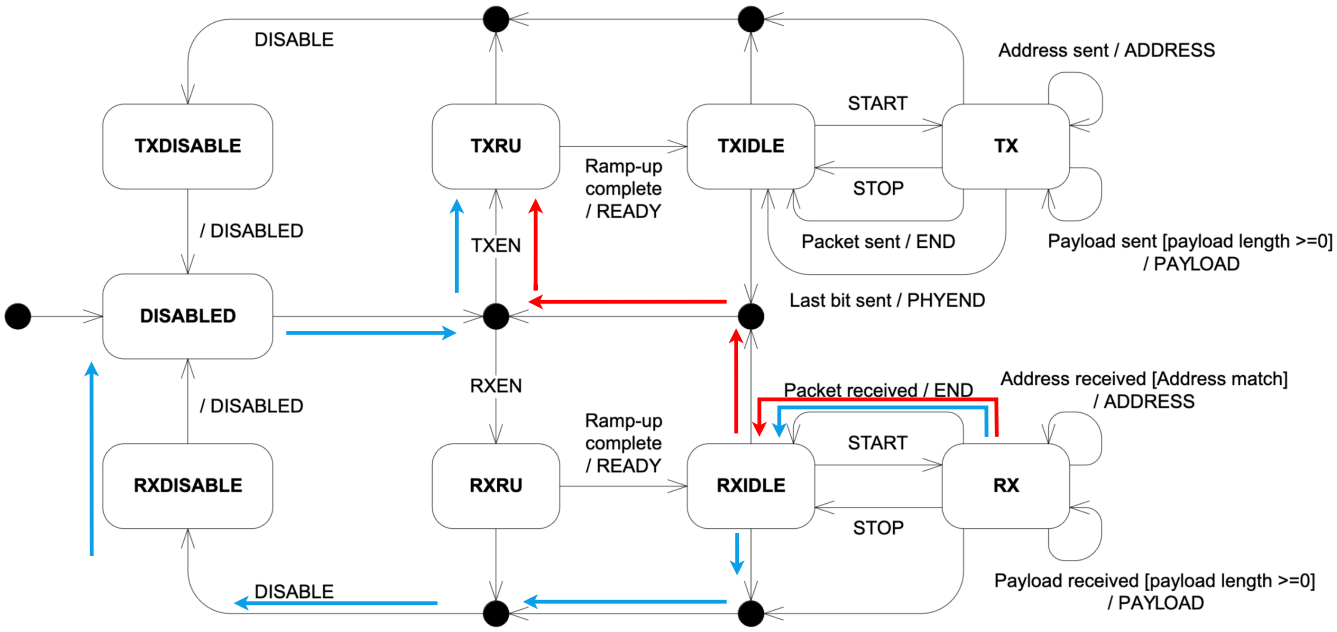

Figure 14. The normal RX to TX switching procedure is highlighted with blue arrows, according to nRF52840 Product Specification v1.7 Section 6.20.5. In contrast, the undocumented feature implemented by *VaktBLE* is denoted with red arrows. In summary, the undocumented feature transitions the state of the radio from **RXIDLE** to **TXRU**, thus reaching state **TX** in a shorter path.