Recommended Paper

Breaktooth: Breaking Security and Privacy in Bluetooth Power-Saving Mode

Keiichiro Kimura[1,a)] Hiroki Kuzuno[1,b)] Yoshiaki Shiraishi[1,c)] Masakatu Morii[1,d)]

Received: December 6, 2024, Accepted: xx xx, xxxx

Abstract: With the increasing demand for Bluetooth devices, various Bluetooth devices support a power-saving mode to reduce power consumption. One of the features of the power-saving mode is that the Bluetooth sessions among devices are temporarily disconnected or are close to being disconnected. Prior works have analyzed that the power-saving mode is vulnerable to denial of sleep (DoSL) attacks that interfere with the transition to the power-saving mode of Bluetooth devices, thereby increasing its power consumption. However, to the best of our knowledge, no prior work has analyzed vulnerabilities or attacks on the state after transitioning to the power-saving mode. To address this issue, we present an attack that abuses two novel vulnerabilities in sleep mode, which is one of the Bluetooth power-saving modes, to break Bluetooth sessions. We name the attack Breaktooth. The attack is the first to abuse the vulnerabilities as an entry point to hijack Bluetooth sessions between victims. The attack also allows overwriting the link key between the victims using the hijacked session, enabling arbitrary command injection on the victims. Furthermore, while many prior attacks assume that attackers can forcibly disconnect the Bluetooth session using methods such as jamming to launch their attacks, our attack does not require such assumptions, making it more realistic. In this paper, we present the root causes of the Breaktooth attack and their impact. We also provide the technical details of how attackers can secretly detect the sleep mode of their victims. The attackers can easily recognize the state of the victim's Bluetooth session remotely using a standard Linux command. Additionally, we develop a low-cost toolkit to perform our attack and confirm the effectiveness of our attack. Then, we evaluate the attack on 17 types of commodity Bluetooth keyboards, mice and audio devices that support the sleep mode and show that the attack poses a serious threat to Bluetooth devices supporting the sleep mode. To prevent our attack, we present defenses and their proof-of-concept. We responsibly disclosed our findings to the Bluetooth SIG. We also released the toolkit as open-source.

Keywords: Bluetooth, power-saving mode, session hijack, spoofing, defenses

1. Introduction

Bluetooth is a pervasive technology for low-power, short-range wireless communication. It provides two specifications: Bluetooth Classic and Bluetooth Low Energy (BLE). We focus on Bluetooth Classic, from now on indicated as Bluetooth. Bluetooth is globally adopted as a technology for connecting devices, such as wireless headphones, keyboards, mice, and speakers, to PCs and mobile phones. Bluetooth-equipped devices are becoming increasingly popular [1], [2].

With the increasing demand for Bluetooth devices, many of them support a power-saving mode. This mode is designed to limit background operations of the devices, reducing power consumption. There have been discussions about attacks that abuse Bluetooth power-saving modes [3], [4]. Specifically, the DoSL attacks against devices that implement the power-saving mode have been discussed. The DoSL attacks involve attackers preventing the victim's Bluetooth device from transitioning to the power-saving mode, thereby increasing the power consumption of the device. However, *no* prior work has discussed vulnerabilities or attacks on the state *after* the transition to the power-saving mode.

In this paper, we present the **Breaktooth** attack, a novel attack that breaks Bluetooth sessions between victims who support Bluetooth sleep mode, one of the power-saving modes. The attack abuses two novel vulnerabilities of the sleep mode we uncover; the first is that the transition to the sleep mode causes the Bluetooth between victims to be silently disconnected without the victim's interactions. The second is that the victim's master transitions to a state in which it accepts connection requests from the victim's slave after the transition to sleep mode. In our attack, the attacker hijacks the Bluetooth session between the victims, aiming at the moment when the victim's slave transitions to sleep mode.

The attack strategy consists of four steps: (1) *impersonate* a victim's slave device, (2) *hijack* the Bluetooth session between the victim master device and the slave abusing the sleep mode vulnerabilities and establish connections with the master, (3) *overwrite* a link key that is already shared between the two victims and generate a new link key with the master by downgrading the security level, and (4) *inject* arbitrary commands into the master using the link key generated in (3). Furthermore, we detail the root causes and their impact, as well as the technical details of

[1] Graduate School of Engineering, Kobe University, Kobe, Hyogo 657–8501, Japan
[a)] k_kimura@stu.kobe-u.ac.jp
[b)] kuzuno@port.kobe-u.ac.jp
[c)] zenmei@port.kobe-u.ac.jp
[d)] mmorii@kobe-u.ac.jp

how attackers can remotely detect the sleep mode between the victims.

We develop and release a low-cost toolkit to perform our attack [5] and evaluate the attack against commodity Bluetooth devices using the toolkit. We evaluate our attack on 17 unique commodity Bluetooth devices (e.g., keyboards, mice and audio devices) as the slave and three unique devices (e.g., laptops and smartphones) as the master. We have *successfully* exploited a broad set of operating systems (e.g., Windows, iOS, and Android) and vendors (e.g., Ewin, ELECOM, Buffalo, iClever, and Ancker). Based on our evaluation results, we discuss the threat of the Breaktooth attack and its comparison with prior attacks. Moreover, we discuss defenses against our attack.

We summarize our main contributions as follows:

- We uncover *two novel* vulnerabilities of Bluetooth sleep mode and present the Breaktooth attack that abuses these vulnerabilities. The vulnerabilities allow attackers to hijack the Bluetooth session between the victim's master and slave without special privileges or tools. Our attack is the first to hijack a Bluetooth session while the victim is in sleep mode.
- We present the details of the Breaktooth attack root causes and their impact. Owing to the root causes, we discuss the impact of the attack, which makes many prior attacks even *more realistic*. We describe the technical details of how attackers *secretly* detect the sleep mode between victims.
- We release a *low-cost* toolkit to reproduce our attack. The toolkit supports functions to reproduce our attack, such as spoofing, sleep mode detection, overwriting a pre-shared link key between the victims, and injecting arbitrary commands to the victims. Our toolkit complements the state-of-the-art Bluetooth security testing, such as [6], [7], [8].
- We evaluate our attack on 17 unique commodity devices, including keyboards, mice and audio devices supporting the sleep mode. The attack is *successful* against *all* 17 devices and demonstrates that our attack enables attackers to hijack Bluetooth sessions, overwrite the victim's link key, and inject arbitrary commands into the victims. From the evaluation results, we confirm that the Breaktooth attack is *practical*, and we discuss its threats and practical defenses.

Ethical Considerations and Responsible Disclosure: This work investigates unknown threats to widespread technologies and proposes defenses. All experiments were conducted in-house; no external devices were attacked. We responsibly disclosed our findings to the Bluetooth SIG in May 2024. We also proposed a patch for the findings. They have responded to our report and we are coordinating with them on the timing of the security notification from the Bluetooth SIG.

The remainder of this paper is organized as follows: In Section 2, we briefly introduce Bluetooth and Bluetooth power-saving mode. In Section 3, we present our system and attacker model. In Section 4, we describe the Breaktooth attack. Section 5 presents the implementation of our attack. In Section 6, we evaluate the impact and effectiveness of our attack. In Section 7, we discuss the attack and our proposed defenses. Related work is presented in Section 8. Conclusions are presented in Section 9.

2. Background

2.1 Bluetooth

Bluetooth, a wireless communication technology established by the Bluetooth SIG, is widely used for low-power, short-range wireless communications [9]. Bluetooth operates in the 2.4 GHz Industrial Scientific and Medical band (ISM), with 79 channels spaced 1 MHz apart, and employs Frequency-Hopping Spread Spectrum (FHSS) as its channel access method [9]. FHSS is adopted to provide more reliable communication (e.g., avoiding interference with Wi-Fi). The Bluetooth network, known as piconet, consists of one master device providing a reference CLK clock signal [10] and up to seven slave devices synchronized with the master to form a piconet.

The Bluetooth architecture is divided into the Bluetooth controller and the Bluetooth host [6], [8], [11]. The controller implements the physical layer and link manager in the Bluetooth chip, whereas the host implements Logical Link Control and Adaptation Protocol (L2CAP), Radio Frequency Communication (RFCOMM), and the application layer in the device operating system (OS). The controller and host communicate via the Host Controller Interface (HCI). Bluetooth devices use the Service Discovery Protocol (SDP) to broadcast their service information to other devices [12].

2.2 Bluetooth Security

In Bluetooth, a common key called the link key is utilized for the authentication and encryption of communication between master and slave devices. The link key is generated and shared between the master and the slave during pairing, which takes place over the Link Manager Protocol (LMP) [6], [13]. If the information necessary to identify the link key is leaked to attackers during pairing, the link key can be compromised, leading to potential eavesdropping and communication tampering.

Secure Simple Pairing (SSP) is the most secure and widely used pairing mechanism that prevents link key leakage during pairing [14], [15]. In SSP, the link key is derived from an ECDH shared secret key, which is not transmitted over the communication path, making it difficult for attackers to determine the link key [16], [17].

2.3 Bluetooth Power-Saving Mode

Bluetooth power-saving modes can be broadly categorized into two types: those established by the Bluetooth SIG, namely sniff mode, hold mode, and park mode [18], [19], [20], [21], and the one not established by the Bluetooth SIG, which is sleep mode [3].

Sniff Mode: In this mode, the device increases its listening interval. The slave device transitions into sniff mode upon receiving a sniff command message from either a master device or another slave device. The sniff interval can range from a few seconds to longer periods and is suitable for situations with a long communication lag. However, there is no guarantee that the device will receive maintenance, such as maintaining or checking the Bluetooth connection at each sniff interval. The power efficiency of the device in the sniff mode is lower than that in the hold or park

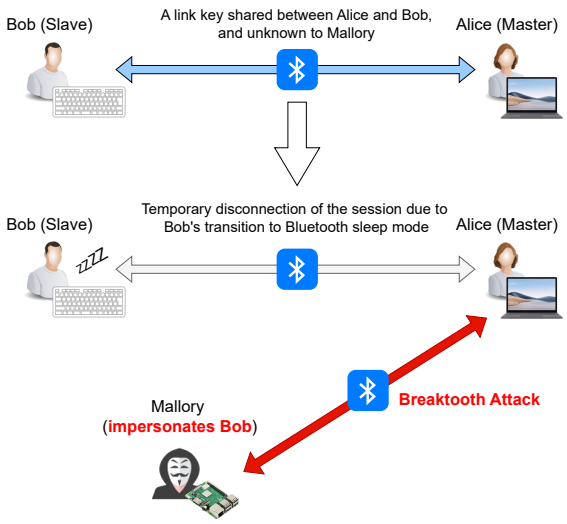

Fig. 1 **Threat model for Breaktooth attacks**: Alice and Bob share a link key that is unknown to Mallory. Mallory, impersonating Bob, aims to establish Bluetooth sessions with Alice by abusing Bob's Bluetooth power-saving mode. Mallory then overwrites the link key and injects arbitrary commands into Alice via the established sessions.

modes [22], [23].

Hold Mode: Hold mode enhances power efficiency by transitioning the device into a short-term inactive state. Only an internal timer called '`holdTO`' operates in this mode, and data transfer resumes once it expires. The device can independently transition into hold mode either through the master or by requesting the master as a slave. The inactive duration is agreed upon by the master and slave beforehand. The hold mode's power efficiency is higher than that of sniff mode but lower than that of park mode [22], [24].

Park Mode: When a slave device maintains an inactive state within a piconet for a relatively longer period compared to the sniff or hold mode, it transitions to the park mode. The slave in park mode maintains synchronization with the piconet while remaining uninvolved in traffic. To return from the park mode, the slave must request and be granted a transition to the active mode by the master. The power efficiency of the device in the park mode is higher than that in both the sniff and hold modes [22], [25]. Park mode has been deprecated since Bluetooth version 5.0 [26], [27].

Sleep Mode: This mode minimizes power consumption by completely shutting down most or all communication and other functionalities for devices. The Radio Frequency (RF) module is entirely turned off or operated at a low-power consumption level. Therefore, resynchronization with the piconet may take some time when a Bluetooth device returns from sleep mode. The sleep mode is suitable for long periods of inactivity and is employed in situations where a maximum reduction in power consumption is required. It is implemented in devices such as Bluetooth keyboards and mice, which are battery-powered and operated via Bluetooth.

3. Threat Model

In this section, we define our system and attacker models (**Figure 1**), as well as the notation we use in the rest of the paper.

3.1 System Model

We consider Alice and Bob (i.e., the victims), who are securely communicating via Bluetooth. Alice and Bob represent arbitrary devices (e.g., laptops, smartphones, keyboards, and mice) and can employ any Bluetooth profile (e.g., Human Interface Device Profile (HID), Advanced Audio Distribution Profile (A2DP), and A/V Remote Control Profile (AVRCP)). We assume the victims have already paired using their strongest security capabilities (e.g., SSP and secure connections) and shared a link key.

Without loss of generality, we assume that Alice is a Bluetooth master device (e.g., laptops and smartphones), and Bob is a Bluetooth slave device (e.g., keyboards, mice, and audio devices). The paired victims have established secure connections using the shared link key. Furthermore, we assume Bob supports the sleep mode.

3.2 Attacker Model

We assume that Mallory is an attacker. Mallory aims to impersonate Bob, establish secure connections with Alice, and hijack Alice's operations using advanced privileges (e.g., keyboard input) obtained from sensitive profiles (e.g., HID).

Mallory must be physically in the victims' Bluetooth range. Mallory does not observe a secure pairing process between Alice and Bob, nor does she recognize the link key. Mallory can capture unencrypted Bluetooth packets and recognize public information about the victims such as Bluetooth names and addresses [13], [28], [29]. Mallory monitors the state of the victim Bluetooth session (Section 4.2 for technical details) and waits until the session is disconnected.

3.3 Notation

In this paper, we use the following notation. We indicate the link key shared between Alice and Bob as LK_{AB} and the link key shared between Mallory and Alice as LK_{MA}. Furthermore, we abbreviate Alice's Bluetooth name as $BTNAME\_A$ and her Bluetooth address as $BTADDR\_A$. Similarly, we abbreviate Bob's Bluetooth name as $BTNAME\_B$ and his Bluetooth address as $BTADDR\_B$.

4. The Breaktooth Attack

In this section, we introduce the Breaktooth attack, which targets any device supporting Bluetooth sleep mode. The attack strategy consists of the following four steps (***Step#1*** to ***#4***):

Step#1. Spoofing Mallory changes her Bluetooth name and its address to impersonate Bob.

Step#2. Session Hijack Mallory, impersonating Bob in ***Step#1***, detects the temporary disconnection state between Alice and Bob caused by the sleep mode, and at this moment sends a connection request to Alice as Bob, hijacking the Bluetooth session between Alice and Bob.

Step#3. Link Key Hijack After ***Step#2***, Mallory sends a pairing request to Alice while impersonating Bob, generating a new link key between them (LK_{MA}). This invalidates LK_{AB}. Mallory does this by bypassing the PIN code authentication.

Step#4. Command Injection By abusing the hijacked link key

in ***Step#3***, Mallory controls Alice's operations using sensitive profiles. For example, Mallory uses the HID profile to inject malicious commands into Alice.

In particular, the *novelty* of the Breaktooth attack lies in the fact that it abuses *unknown* vulnerabilities in the Bluetooth sleep mode to *hijack* the victim's Bluetooth session (***Step#2***), which is the very root cause of the Breaktooth attack.

In this section, we first present the root causes of Breaktooth and their impact. Then, we present the technical details of how Mallory detects the sleep mode of the victims and the details of our attack strategy.

4.1 Root Causes

4.1.1 Sleep Mode Vulnerabilities

The root causes of Breaktooth are the following *two novel* vulnerabilities (***Vuln.#1*** and ***#2***) in the Bluetooth sleep mode.

***Vuln.#1*: Silent Bluetooth disconnection without the victim's interactions:** If the Bluetooth session between Alice and Bob is inactive for more than a certain period, Bob sends a disconnection request to Alice. Alice accepts the request and temporarily disconnects the Bluetooth between them. The disconnection is performed *silently*, *without* user (victim) interaction, and *without* notification to the user. Therefore, recognizing the disconnection is *difficult* for the user.

Vuln.#2*: Alice's transition to an acceptance state for connection requests from Bob after the sleep mode:** After the disconnection between Alice and Bob due to the sleep mode (Vuln.#1***), Alice *accepts* connection requests from Bob. If Mallory impersonates Bob, but Alice and Bob have established a Bluetooth session and are communicating, Alice does not accept the connection request from Mallory. However, if the Bluetooth session between the victims is temporarily disconnected due to the sleep mode, Alice assumes that Bob will return from the sleep mode and transitions to a state where she accepts Bluetooth connection requests from Bob. Therefore, Mallory, as Bob, can establish a Bluetooth connection with Alice.

By abusing these vulnerabilities, Mallory can *easily* hijack the Bluetooth session between Alice and Bob. **Figure 2** illustrates a scenario in which Mallory hijacks the Bluetooth session between the victims by abusing the vulnerabilities in sleep mode.

Alice and Bob are communicating via Bluetooth. However, if Bob remains inactive for more than a certain period, he sends a disconnection request to Alice to transition to sleep mode. Alice accepts this request, and the Bluetooth between Alice and Bob is temporarily disconnected (***Vuln.#1***). After the disconnection, Alice transitions to the state that accepts connection requests from Bob (***Vuln.#2***).

Abusing the vulnerabilities, Mallory impersonates Bob and sends a connection request to Alice. Mistaking Mallory for Bob, Alice accepts the request, temporarily establishing a connection with Mallory. As a result, legitimate Bob can *not* restore the Bluetooth connection with Alice, as it has been hijacked by Mallory.

4.1.2 Root Causes Impact

The root causes make prior attacks on Bluetooth (e.g., [7], [10], [28], [30], [31], and [32]) even *more realistic* and have a *severe* impact on Bluetooth security.

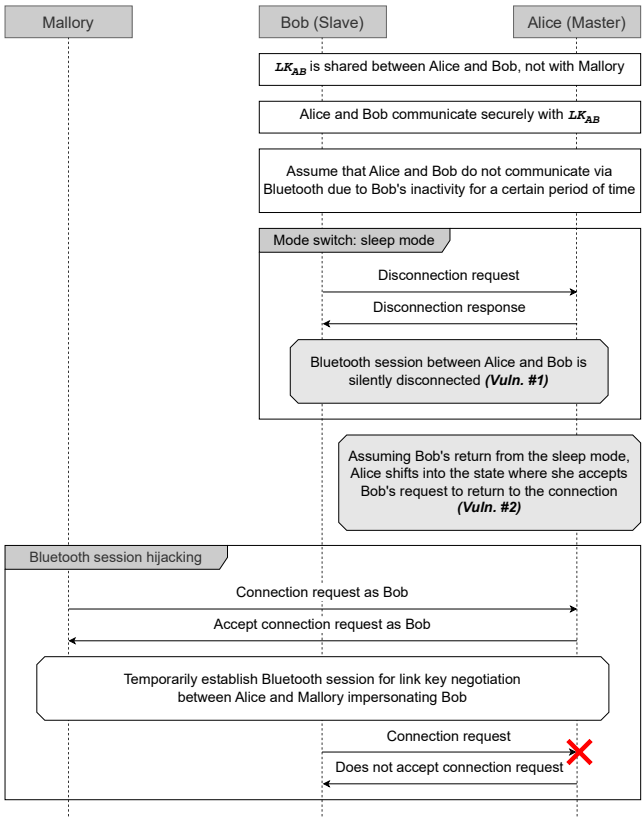

Fig. 2 **The Breaktooth attack root causes and session hijack scenario abusing them**: Assume that Alice and Bob have already paired and share LK_{AB}. If Bob remains inactive for a certain period, Bob transitions to the sleep mode and the session between Alice and Bob is silently disconnected (***Vuln.#1***), and Alice becomes ready to accept a connection from Bob (***Vuln.#2***). After ***Vuln.#2***, Mallory impersonates Bob and sends a connection request to Alice. Alice accepts the request, and Mallory temporarily establishes a connection with Alice. Because Mallory has hijacked Bob's connection with Alice, Bob cannot restore the connection with Alice.

Many prior attacks require attackers to forcibly disconnect the Bluetooth session among the victim's devices to launch their attacks. For example, BIAS [7], Blacktooth [28], and Key Negotiation Downgrade Attacks on Bluetooth BR/EDR and BLE [10] define their attacker models that involve jamming the Bluetooth spectrum among the victims to disconnect their session. However, these attacks do not discuss the methods and practicality of jamming to forcibly disconnect Bluetooth sessions. The downgrade attacks proposed by Zhang et al. [32] also needs jamming to launching the attack. We argue that such assumptions are *strong* and *impractical*.

The sleep mode vulnerabilities lead to a temporary disconnection of Bluetooth *without* the attacker's intervention and the victim's interaction. This state provides attackers with a starting point for launching attacks *without* the need to forcibly disconnect the Bluetooth session among victims, such as by using jamming tools. By abusing these vulnerabilities, the prerequisite for forcible disconnection is removed, making the prior attacks considerably more realistic.

4.2 How Attackers Detect the Sleep Mode

Mallory must *secretly* detect whether the Bluetooth session be-

tween Alice and Bob has transitioned to the sleep mode. Mallory can *easily* detect this using `l2ping`. `l2ping` is a command used in Linux systems that sends an L2CAP echo request to the Bluetooth address specified in dotted hexadecimal notation [33], [34]. To the best of our knowledge, *no* prior works have investigated the effectiveness of using the `l2ping` command as a method to secretly detect the sleep mode among victims remotely.

In this section, we first describe three patterns of response behavior to `l2ping` echo requests. Subsequently, we present the technical details of how Mallory uses the response behavior to detect the sleep mode.

4.2.1 L2ping Echo Response Behavior

We describe three patterns of echo response behavior that Mallory receives from Alice when Mallory sends a `l2ping` echo request to Alice. For simplicity, we notate the states of Alice, Bob, and Mallory as follows:

- **C(A–B)**: Alice and Bob have established a Bluetooth session and are active.
- **NC(A–B)**: The Bluetooth session between Alice and Bob is disconnected.
- **M(B)**: Mallory impersonates Bob.
- **M**: Mallory does not impersonate Bob.

According to the above notation, we describe the behavior of the following three `l2ping` echo response behavior, ***Behaviors#1, #2***, and ***#3*** (**Figure 3**).

***Behavior#1*: C(A–B) and M:** Even when Alice and Bob have established a Bluetooth session, if Alice's Bluetooth is active, Mallory sends a `l2ping` echo request to Alice, and Alice sends an echo response to the request. This is standard `l2ping` behavior. Even if Alice and Bob are not connected via Bluetooth, if Alice's Bluetooth is active, Alice will respond to the echo request from Mallory.

***Behavior#2*: C(A–B) and M(B):** Consider the scenario where Alice and Bob have already established a Bluetooth session, and Mallory impersonates Bob. In this scenario, even if Mallory sends an `l2ping` echo request to Alice, Alice does not respond. The reason is that Alice and the legitimate Bob have already established the session, and Alice recognizes the session with Bob. Therefore, Mallory does not receive a response from Alice, and will get an error (e.g., "Host is down" and "Operation in progress").

Behavior#3*: NC(A–B) and M(B):** Consider the scenario where Alice and Bob have disconnected their Bluetooth session, and Mallory impersonates Bob. In this scenario, if Mallory sends a `l2ping` echo request to Alice, Alice responds. Compared with ***Behavior#2, Alice is not connected to the legitimate Bob via Bluetooth, and she does not recognize the session with Bob. Resultantly, Alice sends the echo response to Mallory, impersonating Bob.

4.2.2 Technical Details to Detect the Sleep Mode

Figure 4 shows the technical details of how Mallory *secretly* detects that the Bluetooth session between Alice and Bob has transitioned to the sleep mode. For detection, ***Behaviors#2*** and ***#3*** described in Section 4.2.1 are used. Mallory sends `l2ping` echo requests intermittently (e.g., at one-second intervals) and checks the response from Alice to detect the state of the Bluetooth session between Alice and Bob.

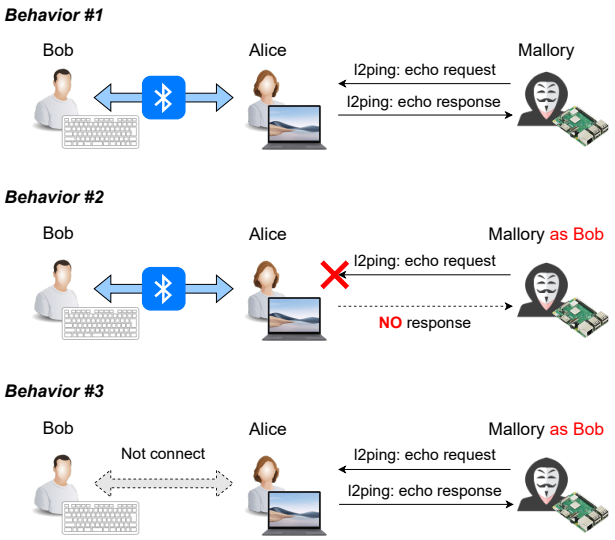

Fig. 3 **Three patterns of `l2ping` echo response behaviors**: ***Behavior#1*** occurs when Alice and Bob are connected, and Alice responds to Mallory's `l2ping` echo requests. This is a normal behavior of the `l2ping` command. ***Behavior#2*** occurs when Mallory impersonates Bob while Alice and Bob are connected; Alice ignores Mallory's `l2ping` echo requests. ***Behavior#3*** occurs when Alice and Bob are disconnected; Alice responds to Mallory's `l2ping` echo requests even if Mallory impersonates Bob.

From ***Behavior#2***, while Alice and Bob are communicating, Alice does not respond to `l2ping` echo requests from Mallory, who is impersonating Bob. Even during periods when Bob is inactive, or no data transmission occurs between Alice and Bob, the session between Alice and Bob remains established, and thus, Alice does not respond to requests from Mallory impersonating Bob.

From ***Behavior#3***, after Bluetooth between Alice and Bob is disconnected due to the sleep mode, Mallory, impersonating Bob, sends a `l2ping` echo request to Alice, to which Alice responds. Thus, if Mallory sends `l2ping` requests intermittently and Mallory, impersonating Bob, detects a response from Alice after a certain point, Mallory can recognize that the Bluetooth session between Alice and Bob has been disconnected.

4.3 Details of the Breaktooth Attack Strategy

In our attack, Mallory aims to hijack the Bluetooth session between Alice and Bob, abusing ***Vuln.#1*** and ***#2***, overwrite LK_{AB}, and control Alice's operations. In this section, we describe the details of each step in the attack strategy, ***Step#1*** to ***#4***, of the Breaktooth attack.

4.3.1 Step#1. Spoofing

The first step for every Bluetooth session is inquiring about the device information [28]. Before establishing a Bluetooth session, Alice requests information from Bob, such as his Bluetooth name and capabilities. Mallory can easily change this information to the same as Bob's. If Mallory connects to Alice via Bluetooth using the same name as Bob, Alice will not recognize the anomaly. Concurrently, Mallory changes her Bluetooth address to $BTADDR\_B$. Alice identifies Mallory as Bob because

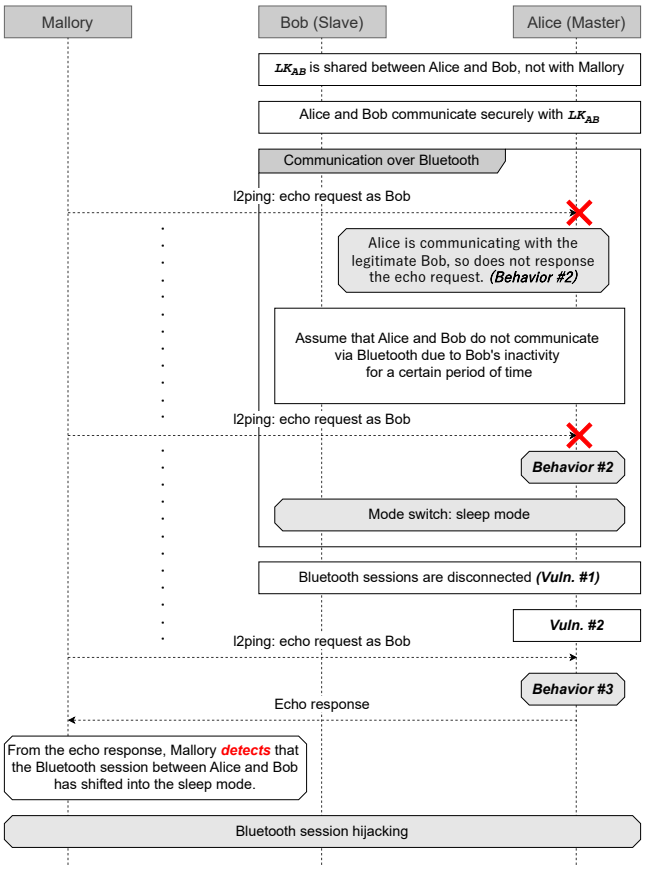

Fig. 4 **Technical details to detect Bluetooth sleep mode**: If Alice and Bob have established a Bluetooth session, when Mallory impersonates Bob and sends a `l2ping` echo request to Alice, Alice does not respond to this request. However, if the session between Alice and Bob is temporarily disconnected due to the sleep mode, when Mallory impersonates Bob and sends a `l2ping` echo request to Alice, Alice responds to it. In this way, Mallory can *secretly* detect the sleep mode between Alice and Bob.

Bluetooth identifies devices using their Bluetooth addresses. The Bluetooth address can also be easily changed [7], [28].

4.3.2 Step#2. Session Hijack

After ***Step#1***, Mallory attempts to hijack the Bluetooth session between Alice and Bob. To achieve this, Mallory abuses the sleep mode vulnerabilities (***Vuln.#1*** and ***#2***) described in Section 4.1, and the detection method for the sleep mode described in Section 4.2.2.

According to Blacktooth [28], some devices do not respond to new inquiry packets after pairing. Assuming a laptop (master) that is paired with a Bluetooth keyboard (slave), if an attacker impersonates the slave and requests the master to connect, the master will not respond to this connection request. The opposite is also true; if an attacker impersonates and requests the slave to connect, the slave will not respond. Therefore, it is *unrealistic* for an attacker to hijack the Bluetooth session between the victim's master and slave that is already paired and communicating. Consequently, prior works [7], [10], [28], [30], [31], [32] have noted the need to forcibly disconnect Bluetooth (e.g., by jamming) before launching their attack.

However, even if the pairing between the victim's master and slave is completed, attackers can actually *still* hijack the Bluetooth session between the victims. The hijacking process follows the same flow as shown in **Fig. 2** and **Fig. 4**. While Alice and Bob communicate via Bluetooth, if Mallory sends `l2ping` echo requests as Bob to Alice, Alice does not respond (***Behavior#2***). However, when the Bluetooth session between the victims transitions to the sleep mode, the connection between the victims is silently disconnected (***Vuln.#1***) and Alice changes her status to accept connection requests from Bob (***Vuln.#2***). After transitioning to sleep mode, if Mallory sends the echo requests as Bob to Alice, Alice responds (***Behavior#3***), and Mallory recognizes that the Bluetooth session between the victims transitioned to sleep mode. After detecting the sleep mode, Mallory sends a connection request as Bob to Alice, which Alice accepts, establishing a new Bluetooth connection (***Vuln.#2***).

4.3.3 Step#3. Link Key Hijack

After ***Step#2***, Mallory overwrites LK_{AB} with LK_{MA} to communicate with Alice.

Mallory requests pairing with Alice. Because Alice misidentifies Mallory for Bob, she accepts the pairing request from Mallory. Mallory sets the IO capability to `NoInputNoOutput` to avoid the PIN authentication [13], [30] before requesting Alice to pair.

When the pairing between Alice and Mallory succeeds, a new link key (LK_{MA}) is generated and shared between them. Because Alice misidentifies Mallory as Bob, Alice overwrites LK_{AB} with LK_{MA}. Therefore, LK_{AB} is invalidated by Mallory, and even if Bob returns from sleep mode to active mode, he *cannot* restore Bluetooth with Alice. Furthermore, even if either Bob or Alice is rebooted, Bluetooth between Alice and Bob will not be restored.

4.3.4 Step#4. Command Injection

After ***Step#3***, Mallory exploits Bob's profile to which Alice has granted access. For example, if Bob is a Bluetooth keyboard, Alice can connect to Bob's HID profile after pairing. Therefore, if Mallory connects to Alice as Bob and communicates using LK_{MA} shared in ***Step#3***, Mallory can exploit Bob's HID profile to send arbitrary commands to Alice.

For example, if Mallory emulates as a Bluetooth keyboard and exploits Bob's HID profile, she can send keyboard commands to Alice. In addition, Mallory can use shortcut keys to control Alice's operations. For example, Windows defines several keyboard shortcuts, including one that can launch PowerShell with administrative privileges. Mallory can use these shortcuts to send arbitrary text, open arbitrary ports for backdoors, or inject malware. Users who can launch PowerShell with administrative privileges can control most operations on the device via PowerShell. Thus, if Mallory launches PowerShell with administrative privileges, she gets full control over Alice's operations.

5. Implementation

In this section, we describe the Breaktooth attack scenario, devices used for the implementation, and the implementation of our toolkit to perform the Breaktooth attack.

5.1 Attack Scenario

The Breaktooth attack scenario involves three devices: a device (e.g., laptop, desktop PC, smartphone, or tablet) as Alice, a commercial Bluetooth keyboard or mouse as Bob, and a Rasp-

Table 1 Specifications of a device used as Mallory

Device	
Device Model	Raspberry Pi 4 Model B
Operating System	Raspberry Pi OS
System	32bit
Debian Version	11 Bullseye
Kernel Version	6.1
Bluetooth	
BlueZ version	5.55
Bluetooth Manufacturer	Cypress Semiconductor
Bluetooth Version	5.0

berry Pi as Mallory. The Raspberry Pi is connected to a wired keyboard or mouse via USB for injecting commands into Alice. The Raspberry Pi not only serves as a spoofing device but also hijacks Bluetooth sessions and LK_{AB} and injects arbitrary commands. Alice and Bob have paired and shared LK_{AB}, which is unknown to Mallory.

5.2 Attack Device

Table 1 lists the specifications of Mallory. Mallory is a Raspberry Pi 4 Model B. The operating system of the Raspberry Pi is the Raspberry Pi OS (11 Bullseye) with Linux OS kernel version 6.1 [35]. The Raspberry Pi OS (11 Bullseye) is preinstalled with BlueZ 5.55 [36]. We utilize `hciconfig` [37] and `hcitool` [38] to command the necessary HCI configuration and the scanning and enumeration of Bluetooth devices, respectively. These two commands are part of the Bluetooth stack provided by BlueZ and are available by default upon installing the OS. Furthermore, the Raspberry Pi supports the Bluetooth adapter by default. Therefore, the hardware and software costs for the Breaktooth attack are *low*.

5.3 Breaktooth Attack Toolkit

We develop the Breaktooth attack toolkit to perform our attack. In this section, we focus on the technical details of ***Step#1*** to ***#4*** in Section 4.3, which are implemented in the toolkit. We release the toolkit as *open-source* on https://breaktooth.dev [5].

5.3.1 Bluetooth Spoofing (Step#1)

We describe how Mallory changes her Bluetooth name and address to impersonate Bob.

The method for changing the Bluetooth name is as follows: First, we create a file named `machine-info` in the `/etc` directory of Mallory's Raspberry Pi. After creating the file, we define a variable as `PRETTY_HOSTNAME` and set $BTNAME\_B$ to the variable. After setting the variable, restart Bluetooth daemon to reflect the change.

We describe the method for changing Mallory's Bluetooth address. We assume that her Bluetooth address is represented in a hexadecimal format, split into six octets, each consisting of eight bits (e.g., `xx:xx:xx:xx:xx:xx`). First, to read the Bluetooth address, we set OpCode Group Field (OGF) [18] as `0x04` and OpCode Command Field (OCF) [18] as `0x009` using `hcitool`'s `cmd` option [38]. Subsequently, we execute the command using `hcitool`'s `cmd` option [38] with OGF as `0x3f` , OCF as `0x001`, and specify the reversed address of Bob's Bluetooth address to set Mallory's Bluetooth address as Bob's. By executing these commands, Mallory can set her Bluetooth address to the same as Bob's. To ensure that the changes are reflected, reset the Bluetooth device settings on the Linux system using `hciconfig`'s `reset` option [37], and then restarting the Bluetooth service.

5.3.2 Implementation of a Sleep Mode Detector (Step#2)

In Section 4.3.2, before attempting to hijack the Bluetooth session between Alice and Bob, Mallory must monitor the Bluetooth connection state between them to recognize whether the connection state is in the sleep mode or not. Based on the technical details presented in Section 4.2.2, we implement a function to detect the sleep mode in Python. We name the function `sleep_detector`.

Listing 1 shows the source code of `sleep_detector`. The function takes Alice's Bluetooth address as its argument. Mallory sends one `l2ping` echo request per second to Alice. If there is a successful response from Alice to the `l2ping` echo request, the status code of the echo request is zero. Therefore, if the status code of the `l2ping` echo request is non-zero, Mallory continues to send an echo request to Alice. By contrast, if the status code is zero, Mallory recognizes that Bluetooth between Alice and Bob is temporarily disconnected due to the sleep mode, exits `sleep_detector` and Mallory sends a connection request to Alice as Bob.

5.3.3 Pairing without PIN Code (Step#3)

Mallory sets her I/O capability to `NoInputNoOutput` to pair Bluetooth with Alice without a PIN code. `NoInputNoOutput` indicates that the device has no capability for user input or output during the connection and pairing process. After setting her I/O capability, Mallory initiates a Bluetooth socket, and sets the socket type to `SOCK_ROW` and the socket protocol to `L2CAP`. Mallory sets the socket security level to `High` and the socket's destination to $BTADDR\_A$. With those settings, Mallory can pair with Alice without the PIN code. We use the Python `socket` library [39] for those implementations.

Mallory can pair with Alice without PIN code authentication, but it is difficult to completely avoid interaction with Alice during Bluetooth pairing. According to BLAP [13], if Alice's (master) Bluetooth version is 5.0 or higher, Alice is mandated to popup Yes/No confirmation on her screen when Bluetooth pairing. However, the popup *only* asks whether users would accept the pairing request or not. Therefore, there is *no* way for the users (Alice) to judge whether the pairing is being performed with legitimate devices (Bob), and Alice will probably accept the pairing, and Mallory can generate and share a link key with Alice.

5.3.4 Bluetooth Device Emulator (Step#4)

We implement Bluetooth device emulator to inject arbitrary commands to Alice. The emulator operates after ***Step#1*** to ***#3*** are successfully completed. We implement the emulator by leveraging the D-Bus feature, which can communicate with BlueZ [40].

We describe the details of the emulator implementation. As keyboard and mouse emulators share the same technical specifications, we exclusively focus on the keyboard emulator for clarity. The implementation procedures are outlined as follows: First, a D-Bus client running on the Raspberry Pi captures commands entered through the physical keyboard connected to Mallory's Raspberry Pi via USB. Then, the D-Bus client sends the com-

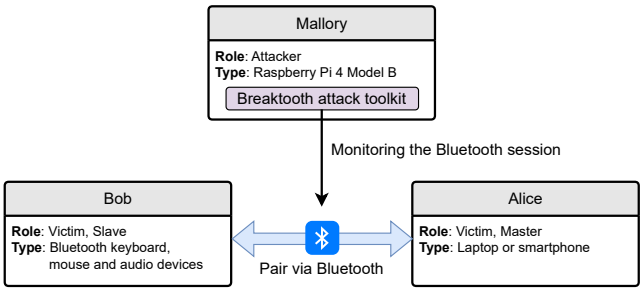

Fig. 5 Experimental setup for Breaktooth attack evaluation

mands to a corresponding D-Bus server, where they are injected into Alice via Bluetooth. We implement the emulator in Python 3.9.2.

D-Bus Service Definition: For emulating Mallory as a Bluetooth keyboard, we define the D-Bus service. In our implementation, we define the D-Bus service as `org.mallory.btkbservice`. After the definition, we set the definition file in `/etc/dbus-1/system.d`, and restart the Bluetooth daemon.

D-Bus Server: We prepare an SDP record file with the settings for a Bluetooth keyboard to emulate Mallory as a Bluetooth keyboard. Then, we register the SDP record file at the HCI path used by Mallory for Bluetooth communication (e.g., `/org/bluez/hci0`). After the SDP record is registered, Mallory is recognized by Alice as an input device by setting the Bluetooth device class to `0x2c0540`. After these settings, we register a method (e.g., `send_keys`) in the D-Bus service, `org.mallory.btkbservice` for transferring keyboard commands, received from the D-Bus client, to Alice. After the method registration, Mallory initiates the D-Bus server and connects to Alice via Bluetooth abusing LK_{MA}. We use `python3-dbus` [41], `python3-bluez` [42], and `python3-gi` [43] for implementation.

D-Bus Client: The D-Bus client must convert the input keys from the wired keyboard into the HID codes [44] and send them to the D-Bus server. The client identifies and tracks events with the `ID_INPUT_KEYBOARD` property from all input events observed on the Raspberry Pi, which refer to events from the wired keyboard. The client also converts keyboard input events into HID codes. After the conversion, the client calls the `send_keys` method registered in the D-Bus service `org.mallory.btkbservice` and specifies the converted HID codes as arguments to send them to the D-Bus server. Subsequently, the D-Bus server transmits the HID codes received from the client to Alice via Bluetooth. We use `python3-evdev` [45], in addition to `python3-dbus` for implementation.

6. Evaluation

In this section, we present our evaluation setup and results.

6.1 Setup

Figure 5 shows the evaluation model of the Breaktooth attack, and **Figure 6** shows the actual environment to evaluate the attack. To evaluate our attack, we first pair Alice and Bob via Bluetooth and establish a Bluetooth session between them. Subsequently, we configure Mallory's Bluetooth name and address to the same as Bob's. With Alice and Bob's Bluetooth session active, we leave Bob without performing any operations. Furthermore, we execute the `sleep_detector` from the Breaktooth attack toolkit, and Mallory secretly monitors the connection status of Alice and Bob's Bluetooth. Finally, we keep Mallory in the monitoring state until Alice and Bob's connection transitions to the sleep mode.

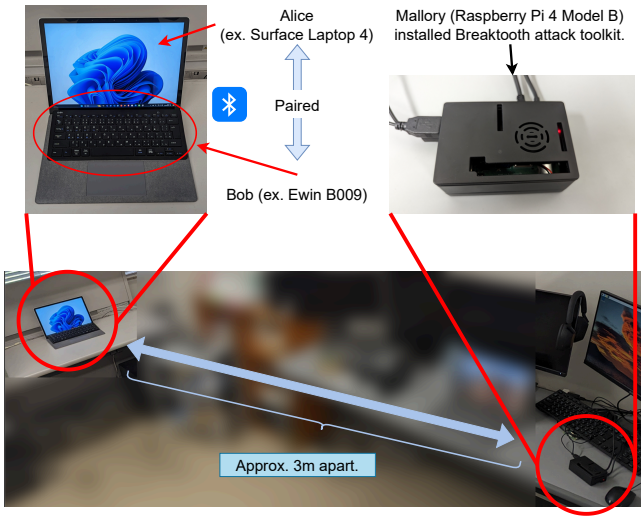

Fig. 6 Evaluation model of the Breaktooth attack

Table 2 **Specifications of devices used as Bob**: K = Keyboard, M = Mouse, A = Audio device, BTV = Bluetooth version, TSM = Time to the sleep mode

Manufacturer	Model	Type	BTV	TSM
Ewin	EW-B009	K	5.1	10 min
Earto	JP-B087-BK	K	5.1	10 min
Buffalo	BSKBB315BK	K	3.0	10 min
SANWA SUPPLY	400-SKB062	K	3.0	10 min
Ajazz	308i	K	3.0	15 min
iClever	IC-BK22	K	5.1	30 min
Anker	A7726	K	3.0	30 min
ELECOM	TK-FBP101WH	K	3.0	30 min
Logicool	K380BK	K	3.0	120 min
Buffalo	BSMBB105BK	M	3.0	10 min
EX-DASH	WM1	M	3.0	10 min
SAMDVM	SAMDVM-13	M	3.0	10 min
Business Harmony	Em23-S1	M	3.0	30 min
Lypertek	Soundfree S10	A	5.2	20 min
Jabra	Elite 7 Active	A	5.2	30 min
Sennheiser	M3IETW	A	5.0	60 min
LIBRATONE	AIR+(2nd)	A	5.2	5 min

* We exclude from our evaluation Bluetooth devices that require a device-specific app to switch to sleep mode.

6.2 Results

We evaluate the Breaktooth attack on three unique commodity devices as Alice (**Table 3**) and 17 unique commodity devices as Bob (**Table 2**). All devices shown in **Table 2** support Bluetooth sleep mode.

Table 4 lists the evaluation results. A white circle (●) indicates that our attack succeeds, and a half-white circle (◐) indicates that the attack succeeds only when the master device's screen is on the Bluetooth pairing screen. A black circle (○) indicates that the attack does not succeed because the transition to power-saving mode cannot be detected.

Table 3 Specifications of devices used as Alice

Manufacturer	Model	Operating System	Driver	Bluetooth Version
Microsoft	Surface Laptop 4	Windows 11 Home	Intel(R) Wireless Bluetooth(R)	5.1
Google	Pixel 2	Android OS 10	-	5.0
Apple	iPhone 11	iOS 16.2	-	5.0

Table 4 Breaktooth attack evaluation results

Bob (Slave)	Alice (Master)		
	Surface Laptop 4	Pixel 2	iPhone 11
EW-B009	●	●	◐
JP-B087-BK	●	●	◐
BSKBB315BK	●	●	◐
400-SKB062	●	●	◐
308i	●	●	◐
IC-BK22	●	●	◐
A7726	●	●	◐
TK-FBP101WH	●	●	◐
K380BK	●	●	◐
BSMBB105BK	●	●	◐
WM1	●	●	◐
SAMDVM-13	●	●	◐
Em23-S1	●	●	◐
Soundfree S10	●	●	◐
Elite 7 Active	●	●	◐
M3IETW	●	●	◐
AIR+(2nd)	○	●	◐

● The attack is successful and the operation of Alice is hijacked by Mallory.

◐ The attack only succeeds when Alice's screen is in the Bluetooth settings.

○ The attack does not succeed because the transition to power-saving mode cannot be detected.

The Breaktooth attack succeeds on all of 17 devices when they are used as Bob. In the case of the Logicool K380BK, although it is not practical as an attack scenario since it takes two hours to transition to the sleep mode, we confirm that the attack succeeds by making the `sleep_detector` (Listing 1) wait for two hours. When Alice is the iPhone 11, our attack succeeds only when Alice's screen is on the Bluetooth pairing screen. When Alice is the Surface Laptop 4 and Bob is the AIR+ (2nd), Mallory fails to hijack the Bluetooth session between Alice and Bob because Alice sends a connection request to Bob immediately after Bob transitions to sleep mode.

In our evaluation, if our attack succeeds, we confirm damages, such as command injection via a terminal launched on Alice with administrator rights. Additionally, if Alice is the Google Pixel 2 or iPhone 11 shown in **Table 3**, we confirm that Mallory can make phone calls on Alice to arbitrary numbers, not only inject commands. When Bob is an audio device, Mallory is confirmed to be able to eavesdrop on audio data and audio metadata being played from Alice. It is also confirmed that during the audio data eavesdropping, the intercepted audio data can be recorded to a `wav` file and played back using the `parecord` command. Furthermore, during our attack, Bob cannot reconnect the Bluetooth with Alice, regardless of the operations performed.

7. Discussion

7.1 Threat Analysis and its Comparisons with Prior Attacks

The Breaktooth attack poses serious threats to all the CIA (Confidentiality, Integrity, and Availability) triad. The specific threats to each of the CIA triad posed by the attack are as follows:

Confidentiality: Access to a victim's master device should be permitted *only* for the victim's slave device. However, the Breaktooth attack enables attackers to access the master device via Bluetooth. Moreover, the attack has demonstrated that the attackers can illegitimately and effortlessly gain administrative rights on the victim's master device, without pre-existing access. Therefore, the Breaktooth attack constitutes a grave threat to confidentiality.

Integrity: If the Breaktooth attack succeeds, the victim's slave device *cannot* transmit commands via Bluetooth correctly to the victim's master device. This incorrect transmission of commands results from inconsistencies in the link key used for encrypting communication between the victims. If the intended commands are not accurately transmitted, the attack compromises integrity.

Availability: Even if the victim's slave device recovers from the sleep mode or is rebooted, it *cannot* communicate normally with the victim's master device via Bluetooth. This inability to conduct normal Bluetooth communication arises from the attacker's hijacking of Bluetooth communication with the master device. The attack renders the services, normally accessible through the slave device, unusable, posing threats to their availability.

Based on the above analysis, we conclude that the Breaktooth attack poses serious threats to all the CIA triad.

Table 5 shows that the Breaktooth attack is the first to abuse the Bluetooth sleep mode as a starting point for hijacking the Bluetooth session. In contrast to many prior attacks for which it is assumed that attackers need to jam the Bluetooth channel or forcibly disconnect the Bluetooth among victims [7], [10], [30], our attack does *not* require such an assumption. Furthermore, there is *no* need to pre-install any malicious applications on the victims.

7.2 Defense Against the Breaktooth Attack

We now discuss defenses against the Breaktooth attack.

7.2.1 Design

To prevent Breaktooth, we define a *sleep state* and manage the state. From ***DF#1*** to ***DF#5***, we propose the design of the sleep state.

DF#1. Bob sends a disconnection request with a sleep flag to Alice.

DF#2. Alice accepts the disconnect request, but in Alice's adapter, Alice sets Bob's state to sleep (`sleeped=TRUE`) and manages Bob as still being in the connected state.

DF#3. Alice notifies her user that Bob has transitioned to the

Table 5 **Threat comparison with prior attacks**: The Breaktooth attack is the first to abuse the sleep mode to hijack Bluetooth sessions between the victims. The attack does not require jamming to forcibly disconnect Bluetooth between the victims. C = Confidentiality, I = Integrity, A = Availability

			Attack					
Year	Paper	Target	Phase	C	I	A	Secure Connection	Jamming or Force to Disconnect
2016	Uher [3]	BLE	Power-Saving Mode (Sleep)	○	◐	○	NA	*Not* required
2019	Antonioli. [6]	Classic	Active Mode (Pairing)	●	●	◐	✓	*Not* required
2020	Antonioli. [7]	Classic	Active Mode (Pairing)	●	●	●	✓	Required
2020	Zhang [32]	BLE	Active Mode (Pairing)	●	●	●	✓	Required
2021	Tsch. [30]	BLE	Active Mode (Pairing)	●	●	●	✓	Required
2021	Antonioli. [10]	Classic/BLE	Active Mode (Pairing)	●	●	◐	✓	Required
2022	Ai [28]	Classic	Active Mode (Pairing)	●	●	●	✓	Required
2024	**Breaktooth**	Classic	Power-Saving Mode (Sleep)	●	●	●	✓	*Not* required

Symbols related to the CIA triad
(●): It poses a grave threat. (◐): It poses a partial threat. (○): It poses no threat.

Symbols for Secure Connection
(✓): The attack is possible regardless of the support for secure connections.

sleep mode (patch for ***Vuln.#1***).

DF#4. Alice does not accept any connection or pairing requests from Bob while managing him in a sleep state (patch for ***Vuln.#2***).

DF#5. Bob notifies Alice that he is waking up from the sleep mode. Alice confirms that Bob still holds LK_{AB}, and manages him as active (`sleeped=FALSE`). Subsequently, she accepts a connection request from Bob.

The reason Alice and Bob confirm LK_{AB} verification (i.e., device authentication) in ***DF#5*** is that Alice can verify that Bob returning from the sleep mode is truly the same device (Bob) that was connected before transitioning to the sleep mode. LK_{AB} does not change before and after transitioning to the sleep mode. If the device confirmation using LK_{AB} is not performed, Mallory can impersonate Bob and notify Alice of the return from the sleep mode.

7.2.2 Proof-of-Concept

To verify the effectiveness of the proposed sleep state as the defense, we briefly implement a proof-of-concept (PoC) mitigation for Linux. We select Linux because it is open-source and is employed on multiple Bluetooth devices, such as Android smartphones, embedded devices, and laptops.

Our PoC mitigation works as follows (**Figure 7**): We pair a Raspberry Pi 3 Model B+ (victim, Alice) with a Bluetooth keyboard supporting Bluetooth sleep mode (victim, Bob). When Bob sends a disconnect request to Alice to transition to the sleep mode, Alice sets Bob's state to sleep while keeping his connection valid. Additionally, Alice is implemented such that if a device is in the connected and sleep state when processing connection and pairing requests from other devices, these requests are discarded. Subsequently, using a Raspberry Pi 4 Model B impersonating Bob (attacker, Mallory), we execute the attack, but Mallory fails to hijack the Bluetooth session between Alice and Bob, confirming the attack is mitigated.

In our PoC, we implement ***DF#1, #2***, and ***#4*** to partially verify the attack prevention. However, completely mitigating the attack and ensuring communication stability requires substantial rewriting of the communication stack processing.

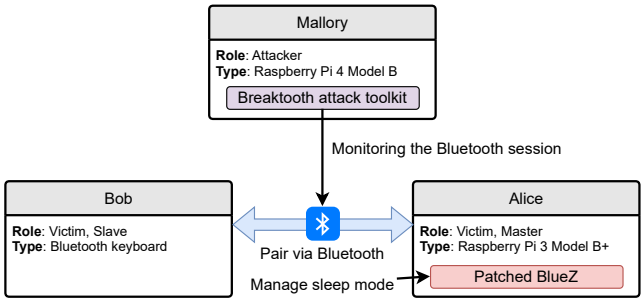

Fig. 7 **PoC model for defense against the Breaktooth attack**: Alice is a Raspberry Pi that implements and installs our proposed defense on BlueZ, the standard Bluetooth protocol stack for Linux. Alice and Bob establish a connection via Bluetooth, and Alice appropriately manages Bob's sleep mode. Meanwhile, Mallory attempts to execute a Breaktooth attack against Alice. However, due to the implemented defense, Mallory fails to hijack the session between Alice and Bob.

7.3 Limitations and Further Attack Scenarios

7.3.1 Extension to BLE

In the attack model shown in **Fig. 2**, our attack fails if Bob employs BLE. Compared with Bluetooth, which allows master/slave role switching, BLE restricts role changes between central (master) and peripheral (slave). Thus, when Mallory, impersonating Bob, sends a connection request to Alice, Alice would recognize Mallory as Bob in the central. However, Mallory fails her spoofing attacks because the legitimate Bob is a peripheral.

The scope of the Breaktooth attack should not only encompass Bluetooth but also be extended to BLE. We believe the issue of spoofing failures can be mitigated by applying prior attack strategies. BLESA [46] is a spoofing attack that exploits vulnerabilities in the BLE reconnection process and has the potential to mitigate spoofing failures. As further attack scenarios, we should apply BLESA in the Breaktooth attack, where Bob employs BLE, to expand the range of our attack to BLE.

7.3.2 Extension to Man-in-the-Middle Attacks

In the attack model shown in **Fig. 2**, Mallory impersonates Bob and establishes Bluetooth sessions with Alice. However, a man-in-the-middle (MitM) attack model where additionally Mallory impersonates Alice and connects to Bob has not been evaluated.

The Blacktooth attack [28] is based on the assumption that the

victim's master and slave are in the Bluetooth connectable state, and the attacker performs a spoofing attack on both the victim's master and slave devices, successfully extending the Blacktooth attack to a MitM attack as the MitM Blacktooth attack. On the other hand, in our evaluation, we find that the device used as Bob (slave, **Table 2**) is no longer in the connectable state after pairing with Alice. Moreover, once the Bluetooth connection with Alice is disconnected, Bob does not accept the connection request from the legitimate Alice, and Bob needs to request a Bluetooth connection himself to Alice to reconnect with Alice. Therefore, the Breaktooth attack is now difficult to extend to a MitM attack such as the MitM Blacktooth attack for the Bob devices evaluated in this paper.

As further attack scenarios, extending the Breaktooth attack to MitM attacks. We will evaluate our attack by extending the types of Bluetooth devices utilized as Bobs and assess the possibility of extending the Breaktooth attack to MitM attacks.

7.3.3 Power-Saving Mode Transition Dependencies

The Breaktooth attack depends on Bob transitioning to sleep mode. If Bob does not transition to sleep mode or remains active, the attack cannot be launched as it relies on the vulnerabilities that emerge during sleep mode transition. While many battery-powered Bluetooth devices implement sleep mode for power conservation, some devices may be configured to never enter sleep mode or have different power-saving behaviors. Additionally, if users frequently interact with Bob (e.g., typing on a keyboard or moving a mouse), the device may not enter sleep mode, preventing the attack. Future work needs to consider the effects of different power-saving behaviors on the success of the Breaktooth attack.

7.3.4 Communication Range Limitations

The success of the Breaktooth attack depends on Bluetooth's communication range limitations. To detect sleep mode transitions and execute the attack, Mallory must maintain stable Bluetooth connectivity with Alice. Standard Bluetooth Classic has an approximate range of 10 meters under ideal conditions, but this range can be significantly reduced by physical obstacles or interference. The attack fails if stable connectivity cannot be maintained due to distance or environmental factors. Future work needs to consider how various environmental conditions and distances affect the success and reproducibility of the Breaktooth attack.

7.3.5 Regional Availability of Evaluation Devices

The Bluetooth devices used in Section 6 include versions 3.0, 5.0, 5.1, and 5.2. This is because, after investigating the available commodity devices supporting the sleep mode in the author's region, only devices with these Bluetooth versions could be obtained. However, we believe that there may be other devices available in regions outside the author's area that support different Bluetooth versions. We have shared this limitation to the Bluetooth SIG, and recognize the need for further investigation to address the limitation. Additionally, we reiterate that the Breaktooth attack does *not* depend on the Bluetooth version. The root causes of the attack are the sleep mode vulnerabilities (***Vuln.#1*** and ***#2***), and we believe that all Bluetooth devices that support the sleep mode are affected by the attack.

8. Related Work

In January 2022, NIST published their latest guidelines on Bluetooth security [47], presenting the security risks associated with Bluetooth, and defenses against them. However, the guidelines do not mention the security risks associated with the Bluetooth power-saving mode. Moreover, few related works have directly discussed the vulnerabilities of the power-saving mode. Research on attacks that abuse the power-saving mode primarily proposes DoSL attacks, interfering with the transition to the power-saving mode.

In 2016, Uher et al. studied DoSL attacks on the BLE protocol and their impacts [3]. BLE sensor nodes adopt a machine-to-machine connection method using the Zero-interaction Authentication (ZIA) model [48]. In the connection method, recognizing whether the connecting device is friendly or hostile is difficult. Uher et al. proposed a DoSL attack that abuses the difficulty. The DoSL attack repeatedly requests unnecessary connections from the BLE sensor devices, reducing their sleep periods and increasing their active periods. As a result, the DoSL attack successfully reduced the theoretical battery life by 93% when the sleep mode of the BLE sensor device was completely eliminated. However, the DoSL attack does not affect confidentiality; therefore, its threat is limited.

In 2018, Huang et al. discussed vulnerabilities of the sniff and park modes and DoS attacks against these modes [4]. In the sniff mode, the master device can only send packets at specific time slots to transition to the sniff mode. To transition to the sniff mode, the master must send a sniff request through the LMP and negotiate parameters with the slave. However, if the slave provides incorrect parameters during negotiation, the master cannot transition to the sniff mode. In the park mode, when the master requests the slave to transition to the park mode, if the slave refuses, neither can it transition to the park mode. Moreover, during the return process from the park mode, if the appropriate response does not reach the master, park commands are issued until the appropriate response returns from the slave. The transition to power-saving mode is hindered if these park commands are disrupted by attackers. However, similar to the work by Uher [3], the main threat is the increase in power consumption, and their threat is limited.

Compared with prior attacks [3], [4], the Breaktooth attack *waits* for the victims to transition to the sleep mode. Then, it abuses the temporary disconnection between the victims to hijack not just the Bluetooth session but most device operations. The attack threatens all aspects of the CIA triad, posing far more serious threats than the prior attacks [3], [4].

9. Conclusion

This paper presents the Breaktooth attack against Bluetooth Classic. The attack is the *first* to abuse the sleep mode vulnerabilities to hijack sessions. It abuses *two novel* vulnerabilities of Bluetooth sleep mode: (1) the silent disconnection of Bluetooth between the victim's master and slave, and (2) after the disconnection, the master becomes receptive to connection requests from the slave. These vulnerabilities enable attackers to *hijack*

Bluetooth sessions between the two victims, *without* needing any preinstalled malicious agent, prior knowledge of the link key, special privileges, and any specialized tool.

The Breaktooth attack is *practical* because it removes the step of forcibly disconnecting the Bluetooth session among victims to launch attacks. Many prior attacks require this step to launch their attack [7], [10], [28], [30], [31], [32]. Our attack abuses Bluetooth's sleep mode to bypass this step *completely*. The sleep mode has been implemented in many devices, especially battery-operated devices such as keyboards, mice and audio devices, adopting the sleep mode for power conservation. By abusing this mode, the Breaktooth attack can target *all* Bluetooth devices that support the sleep mode.

We develop a *low-cost* and reproducible toolkit to perform our attack to demonstrate its feasibility. We use the toolkit to confirm that the Breaktooth attack is effective and impactful. We exploit 17 unique devices, including keyboards, mice, and audio devices supporting the sleep mode. We confirm that the attack succeeds on *all* the devices, allowing the attacker to hijack the sessions and control most operations of the master (e.g., a laptop and a smartphone). In addition, we present defenses against our attack. We propose a sleep state and its management to prevent the attack. We *responsibly disclosed* our findings to the Bluetooth SIG.

Acknowledgments This work is in part conducted under the "Research and development on new generation cryptography for secure wireless communication services" contract for the "Research and Development for Expansion of Radio Wave Resources (JPJ000254)", which is supported by the Ministry of Internal Affairs and Communications, Japan.

References

[1] SIG, B.: 2023 Bluetooth® Market Update, https://www.bluetooth.com/2023-market-update/ (2023). Accessed: 2024-11-05.

[2] SIG, B.: 2024 Bluetooth® Market Update, https://www.bluetooth.com/2024-market-update/ (2024). Accessed: 2024-11-05.

[3] Uher, J., Mennecke, R. G. and Farroha, B. S.: Denial of Sleep attacks in Bluetooth Low Energy wireless sensor networks, *MILCOM 2016 - 2016 IEEE Military Communications Conference*, Baltimore, Maryland, IEEE, pp. 1231–1236 (online), DOI: 10.1109/MILCOM.2016.7795499 (2016).

[4] Huang, Y., Hong, P. and Yu, B.: Design of Bluetooth DOS Attacks Detection and Defense Mechanism, *2018 IEEE 4th International Conference on Computer and Communications (ICCC)*, Chengdu, China, IEEE, pp. 1382–1387 (online), DOI: 10.1109/CompComm.2018.8780851 (2018).

[5] Kimura, K.: Breaktooth, https://breaktooth.dev/ (2024). Accessed: 2024-11-13.

[6] Antonioli, D., Tippenhauer, N. O. and Rasmussen, K. B.: The KNOB is Broken: Exploiting Low Entropy in the Encryption Key Negotiation Of Bluetooth BR/EDR, *28th USENIX Security Symposium (USENIX Security 19)*, Santa Clara, CA, USENIX Association, pp. 1047–1061 (online), available from ⟨https://www.usenix.org/conference/usenixsecurity19/presentation/antonioli⟩ (2019).

[7] Antonioli, D., Tippenhauer, N. O. and Rasmussen, K.: BIAS: Bluetooth Impersonation AttackS, *2020 IEEE Symposium on Security and Privacy (SP)*, San Francisco, CA, USA, IEEE, pp. 549–562 (online), DOI: 10.1109/SP40000.2020.00093 (2020).

[8] Mantz, D., Classen, J., Schulz, M. and Hollick, M.: InternalBlue - Bluetooth Binary Patching and Experimentation Framework, *Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '19, New York, NY, USA, Association for Computing Machinery, p. 79–90 (online), DOI: 10.1145/3307334.3326089 (2019).

[9] SIG, B.: LEARN ABOUT BLUETOOTH: Bluetooth Technology Overview, https://www.bluetooth.com/learn-about-bluetooth/tech-overview/ (2023). Accessed: 2024-11-05.

[10] Antonioli, D., Tippenhauer, N. O. and Rasmussen, K.: Key Negotiation Downgrade Attacks on Bluetooth and Bluetooth Low Energy, *ACM Trans. Priv. Secur.*, Vol. 23, No. 3 (online), DOI: 10.1145/3394497 (2020).

[11] Garbelini, M. E., Bedi, V., Chattopadhyay, S., Sun, S. and Kurniawan, E.: BrakTooth: Causing Havoc on Bluetooth Link Manager via Directed Fuzzing, *31st USENIX Security Symposium (USENIX Security 22)*, Boston, MA, USENIX Association, pp. 1025–1042 (online), available from ⟨https://www.usenix.org/conference/usenixsecurity22/presentation/garbelini⟩ (2022).

[12] AEMIS: BlueBorne White Paper: The dangers of Bluetooth implementations: Unveiling zero day vulnerabilities and security flaws in modern Bluetooth stacks, https://media.armis.com/PDFs/wp-blueborne-bluetooth-vulnerabilities-en.pdf (2023). Accessed: 2024-11-05.

[13] Changseok Koh, Jonghoon Kwon, J. H.: BLAP: Bluetooth Link Key Extraction and Page Blocking Attacks, *2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, Baltimore, Maryland, Dependable Systems and Networks, pp. 227–238 (2022). Available at https://www.computer.org/csdl/proceedings-article/dsn/2022/169300a227/1Fixhx3vELC.

[14] Haataja, K. and Toivanen, P.: Two practical man-in-the-middle attacks on Bluetooth secure simple pairing and countermeasures, *IEEE Transactions on Wireless Communications*, Vol. 9, No. 1, pp. 384–392 (online), DOI: 10.1109/TWC.2010.01.090935 (2010).

[15] Notes, E. B. E.: Secure Simple Pairing Explained, https://www.ellisys.com/technology/een_bt07.pdf (2011). Accessed: 2024-11-05.

[16] Biham, E. and Neumann, L.: Breaking the Bluetooth Pairing – The Fixed Coordinate Invalid Curve Attack, *Selected Areas in Cryptography – SAC 2019: 26th International Conference, Waterloo, ON, Canada, August 12–16, 2019, Revised Selected Papers*, Berlin, Heidelberg, Springer-Verlag, p. 250–273 (online), DOI: $10.1007/978-3-030-38471-5_11$ (2019).

[17] Sun, D.-Z., Mu, Y. and Susilo, W.: Man-in-the-Middle Attacks on Secure Simple Pairing in Bluetooth Standard V5.0 and Its Countermeasure, *Personal Ubiquitous Comput.*, Vol. 22, No. 1, p. 55–67 (online), DOI: 10.1007/s00779-017-1081-6 (2018).

[18] SIG, B.: Bluetooth Core Specification v5.4, https://www.bluetooth.org/DocMan/handlers/DownloadDoc.ashx?doc_id=556599 (2023). Accessed: 2024-11-05.

[19] Conti, M. and Moretti, D.: System level analysis of the Bluetooth standard, *Design, Automation and Test in Europe*, Munich, Germany, IEEE, pp. 118–123 Vol. 3 (online), DOI: 10.1109/DATE.2005.288 (2005).

[20] Willingham, T., Henderson, C., Kiel, B., Haque, M. S. and Atkison, T.: Testing Vulnerabilities in Bluetooth Low Energy, *Proceedings of the ACMSE 2018 Conference*, ACMSE '18, New York, NY, USA, Association for Computing Machinery, (online), DOI: 10.1145/3190645.3190693 (2018).

[21] Satam, S., Satam, P. and Hariri, S.: Multi-level Bluetooth Intrusion Detection System, *2020 IEEE/ACS 17th International Conference on Computer Systems and Applications (AICCSA)*, Antalya, Turkey, IEEE, pp. 1–8 (online), DOI: 10.1109/AICCSA50499.2020.9316514 (2020).

[22] GeekdforGeeks: Modes of Connection Bluetooth, https://www.geeksforgeeks.org/modes-of-connection-bluetooth/ (2023). Accessed: 2024-11-05.

[23] Lin, T.-Y. and Tseng, Y.-C.: An adaptive sniff scheduling scheme for power saving in Bluetooth, *IEEE Wireless Communications*, Vol. 9, No. 6, pp. 92–103 (online), DOI: 10.1109/MWC.2002.1160087 (2002).

[24] Hsu, C.-F. and Hsu, S.-M.: A novel hold-mode-based adaptive interpiconet scheduling algorithm in bluetooth scatternets, *IEEE Wireless Communications and Networking Conference, 2006. WCNC 2006.*, Vol. 1, pp. 469–474 (online), DOI: 10.1109/WCNC.2006.1683509 (2006).

[25] Gonzalez-Castano, F., Gil-Castineira, F. and Garcia-Reinoso, J.: Bluetooth real-time mobile auctions, *2005 IEEE 16th International Symposium on Personal, Indoor and Mobile Radio Communications*, Vol. 3, pp. 2024–2028 Vol. 3 (online), DOI: 10.1109/PIMRC.2005.1651795 (2005).

[26] SIG, B.: Bluetooth Core Specification v4.2, https://www.bluetooth.org/docman/handlers/downloaddoc.ashx?doc_id=441541 (2014). Accessed: 2024-11-05.

[27] SIG, B.: Bluetooth Core Specification v5.0, https://www.

bluetooth.org/docman/handlers/DownloadDoc.ashx?doc_id=421043 (2016). Accessed: 2024-11-05.

[28] Ai, M., Xue, K., Luo, B., Chen, L., Yu, N., Sun, Q. and Wu, F.: Blacktooth: Breaking through the Defense of Bluetooth in Silence, *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, New York, NY, USA, Association for Computing Machinery, p. 55–68 (online), DOI: 10.1145/3548606.3560668 (2022).

[29] Antonioli, D., Tippenhauer, N. O., Rasmussen, K. and Payer, M.: BLURtooth: Exploiting Cross-Transport Key Derivation in Bluetooth Classic and Bluetooth Low Energy, *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '22, New York, NY, USA, Association for Computing Machinery, p. 196–207 (online), DOI: 10.1145/3488932.3523258 (2022).

[30] Tschirschnitz, M., Peuckert, L., Franzen, F. and Grossklags, J.: Method Confusion Attack on Bluetooth Pairing, *IEEE Symposium on Security and Privacy (Oakland)*, San Francisco, CA, USA, IEEE, pp. 1332–1347 (online), available from ⟨https://www.sec.in.tum.de/i20/publications/method-confusion-attack-on-bluetooth-pairing/@@download/file/conference-proceeding.pdf⟩ (2021).

[31] Antonioli, D. and Payer, M.: On the Insecurity of Vehicles Against Protocol-Level Bluetooth Threats, *2022 IEEE Security and Privacy Workshops (SPW)*, San Francisco, CA, USA, IEEE, pp. 353–362 (online), DOI: 10.1109/SPW54247.2022.9833886 (2022).

[32] Zhang, Y., Weng, J., Dey, R., Jin, Y., Lin, Z. and Fu, X.: Breaking Secure Pairing of Bluetooth Low Energy Using Downgrade Attacks, *29th USENIX Security Symposium (USENIX Security 20)*, Boston, MA, USENIX Association, pp. 37–54 (online), available from ⟨https://www.usenix.org/conference/usenixsecurity20/presentation/zhang-yue⟩ (2020).

[33] man page, L.: l2ping(1), https://linux.die.net/man/1/l2ping (2002-2024). Accessed: 2024-11-05.

[34] Yüksel, T., Aydın, c. and Dalkılıç, G.: Performing DoS Attacks on Bluetooth Devices Paired with Google Home Mini, *SSRN Electronic Journal*, Vol. 18, pp. 53–58 (online), DOI: 10.2139/ssrn.4171322 (2022).

[35] Pi, R.: Operating system images: Raspberry Pi OS (Legacy) with desktop, https://www.raspberrypi.com/software/operating-systems/\#raspberry-pi-os-legacy (2023). Accessed: 2024-11-05.

[36] protocol stack, B. O. L. B.: BlueZ 5.55, http://www.kernel.org/pub/linux/bluetooth/bluez-5.55.tar.xz (2016). Accessed: 2024-11-05.

[37] die.net: hciconfig(8) - Linux man page, https://linux.die.net/man/8/hciconfig (2013). Accessed: 2024-11-05.

[38] die.net: hcitool(1) - Linux man page, https://linux.die.net/man/1/hcitool (2012). Accessed: 2024-11-05.

[39] Foundation, P. S.: socket — Low-level networking interface ¶, https://docs.python.org/3/library/socket.html (2024). Accessed: 2024-11-05.

[40] Dunlap, T., Enck, W. and Reaves, B.: A Study of Application Sandbox Policies in Linux, *Proceedings of the 27th ACM on Symposium on Access Control Models and Technologies*, SACMAT '22, New York, NY, USA, Association for Computing Machinery, p. 19–30 (online), DOI: 10.1145/3532105.3535016 (2022).

[41] contributors, D.-B.: dbus-python: Python bindings for D-Bus, https://dbus.freedesktop.org/doc/dbus-python/ (2018). Accessed: 2024-11-05.

[42] Haung, A. and contributors: PyBluez, https://pybluez.readthedocs.io/en/latest/ (2019). Accessed: 2024-11-05.

[43] GNOME: PyGObject, https://pygobject.readthedocs.io/en/latest/ (2023). Accessed: 2024-11-05.

[44] thanhlev: Bluetooth Keyboard Mouse Emulator on Raspberry Pi, https://github.com/thanhlev/keyboard_mouse_emulate_on_raspberry/blob/master/keyboard/keymap.py (2020). Accessed: 2024-11-05.

[45] Valkov, G.: python-evdev, https://python-evdev.readthedocs.io/en/latest/ (2022). Accessed: 2024-11-05.

[46] Wu, J., Nan, Y., Kumar, V., Tian, D. J., Bianchi, A., Payer, M. and Xu, D.: BLESA: Spoofing Attacks against Reconnections in Bluetooth Low Energy, *14th USENIX Workshop on Offensive Technologies (WOOT 20)*, Boston, MA, USENIX Association, (online), available from ⟨https://www.usenix.org/conference/woot20/presentation/wu⟩ (2020).

[47] NIST: Guide to Bluetooth Security, https://www.nist.gov/publications/guide-bluetooth-security-2 (2022). Accessed: 2024-11-05.

[48] Corner, M. D. and Noble, B. D.: Zero-Interaction Authentication, *Proceedings of the 8th Annual International Conference on Mobile Computing and Networking*, MobiCom '02, New York, NY, USA, Association for Computing Machinery, p. 1–11 (online), DOI: 10.1145/570645.570647 (2002).

Appendix

A.1 The Toolkit Source Code

Listing 1 Mallory's `sleep_detector` function

```
import time
import subprocess
"""
bt_addr: Alice's Bluetooth address (BTADDR_A)
"""
def sleep_detector(bt_addr):
    cmd = ["sudo", "l2ping", "-c", "1", "-f", bt_addr]
    print(f"Send a l2ping echo request to {bt_addr} to
      detect Bluetooth sleep mode.")

    while True:
        ret = subprocess.run(cmd)
        if ret.returncode == 0:
            print("Detect the sleep mode !!")
            break
        print(f"{bt_addr} is not in sleep , try again.")
        time.sleep(1)
```