When Good Becomes Evil: Tracking Bluetooth Low Energy Devices via Allowlist-based Side Channel and Its Countermeasure

Yue Zhang
The Ohio State University
zhang.12047@osu.edu

Zhiqiang Lin
The Ohio State University
zlin@cse.ohio-state.edu

ABSTRACT

Bluetooth Low Energy (BLE) is ubiquitous today. To prevent a BLE device (e.g., a smartphone) from being connected by unknown devices, it uses allowlisting to allow the connectivity from only recognized devices. Unfortunately, we show that this allowlist feature actually introduces a side channel for device tracking, since a device with the allowed list behaves differently even though it has used randomized MAC addresses. Worse even we also find that the current MAC address randomization scheme specified in Bluetooth protocol is flawed, suffering from a replay attack with which an attacker can replay a sniffed MAC address to probe whether a targeted device will respond or not based on its allowlist. We have validated our allowlist-based side channel attacks with 43 BLE peripheral devices, 11 centrals, and 4 development boards, and found none of them once configured with allowlisting is immune to the proposed attacks. We advocate the use of an interval unpredictable, central and peripheral synchronized random MAC address randomization scheme to defeat passive device tracking (introducing 1% power consumption overhead for centrals and 6.75% for peripherals, and 88.49 μs performance overhead for centrals and 94.46 μs for peripherals), and the use of timestamps to derive randomized MAC addresses such that attackers can no longer be able to replay them to defeat active device tracking (introducing 3.04% overhead for peripherals, and 63.58 μs and 20.54 μs performance overhead for centrals and peripherals). We have disclosed our findings to Bluetooth SIG and many other stakeholders in October 2020. Bluetooth SIG assigned CVE-2020-35473 to track this logical-level protocol flaw. Google assigned our findings as a high severity design flaw and awarded us with a bug bounty.

CCS CONCEPTS

• **Security and privacy** → **Security protocols**; **Mobile and wireless security** .

KEYWORDS

Bluetooth Security, Wireless Security; Bluetooth Privacy; Identity Resolution Key (IRK); Side Channel Attacks; Replay Attacks

ACM Reference Format:
Yue Zhang and Zhiqiang Lin. 2022. When Good Becomes Evil: Tracking Bluetooth Low Energy Devices via Allowlist-based Side Channel and Its Countermeasure. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS '22), November 7–11, 2022, Los Angeles, CA, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3548606.3559372

1 INTRODUCTION

Being a short-range wireless communication technology, Bluetooth Low Energy (BLE) is ubiquitous and has been used in numerous applications today such as home entertainment, health care, sports, retail, and even recently digital contact tracing. However, BLE devices are subject to MAC address tracking since any nearby attackers can sniff the Bluetooth packets and associate them with particular devices or even users [1–5]. This is because, when using BLE for communication, a peripheral without being connected will periodically (e.g., every 20 milliseconds [6, 7]) advertise its presence to nearby centrals with an Advertising Indication (i.e., `ADV_IND`) packet [8] along with its MAC address; a nearby central (e.g., a smartphone) will typically respond to the `ADV_IND` packet with a scan request (i.e., `SCAN_REQ`) containing the MAC addresses of both the central and the peripheral [2], to see whether the peripheral is a known device or not. Consequently, an attacker with a sniffer can observe MAC addresses being exchanged between Bluetooth devices for MAC address tracking attacks [1–5].

Bluetooth Special Interest Group (SIG) is certainly aware of MAC address tracking threats, and have specified the use of MAC address randomization using such as *Resolvable Private Address* (RPA) to protect the Bluetooth privacy. In particular, RPA allows paired devices (i.e., two devices have exchanged cryptographic keys) to resolve and recognize the peer device's MAC address using its *Identity Resolution Key* (IRK) [6]. With RPA, a Bluetooth MAC address will be randomized periodically (e.g., every 15 minutes [6]), thereby hindering MAC address tracking attacks from nearby attackers.

Unfortunately, in this paper we show that MAC address tracking is still possible even though it is randomized, particularly when the BLE device enables the "filter accept list" [6] defined by Bluetooth SIG (and we call it allowlist for brevity in this paper), an access control feature used by a vast majority of BLE devices (e.g., Android phones, or iPhones). More specifically, when a Bluetooth device is configured with the allowlist, it behaves differently. For instance, a peripheral would ignore `SCAN_REQ` from unknown devices, and only respond to `SCAN_RSP` for its allowed device; a central may directly connect its allowed peripherals (much like a magnet) once receiving an advertisement packet without any `SCAN_REQ`. Therefore, an attacker can track the sniffed MAC addresses and associate them with specific ones by using a sniffer to passively collect and analyze whether the device responded or not. Although both the two communicating devices can enforce address randomization, there could exist an interval, where one device changes its address but the other one remains the same, which consequently leaves significant footprints for attackers to track the devices based on the old and not simultaneously changed address. This allows the attacker to identify both devices across the randomization time interval (e.g., more than 15 minutes).

While the above passive attacks require an attacker to keep monitoring the advertising packets, which would be quite costly for attackers, we also find attackers do not have to do so if they can replay a sniffed MAC address to actively probe whether a peripheral or a central will respond or not, by exploiting the flaw we discovered in the current RPA randomization algorithm in the Bluetooth specification. In particular, although a randomized address in RPA is generated from a random number and a pre-shared IRK between two paired devices, the current Bluetooth protocol does not specify how the random number should be chosen (other than mentioning that the random number should neither be all 0s nor all 1s at page 2,861 in [9]), and no mechanisms are placed to prevent the reuse of an existing random number. Therefore, while an attacker cannot obtain the IRK, he or she can still track the devices across the randomization time interval by simply collecting the sniffed MAC addresses and replaying them to observe whether the devices are in the allowlist or not.

To demonstrate concretely this new MAC address tracking threat, we present a novel allowlist based side channel attack dubbed Bluetooth Address Tracking (BAT). Similar to other MAC address tracking attacks (e.g., [1–5]), our attacks work against devices during the advertising stage. Particularly, we show that an attacker can either *passively sniff* (§4) the Bluetooth packets to identify the peripherals and the centrals, or *actively replay* (§5) the sniffed MAC addresses of centrals to identify their association, or *actively replay* the sniffed MAC addresses of peripherals to attract known centrals. If the attacker knows the user who is using the identified centrals or the peripherals, an attacker can even use BAT attacks to monitor the user's behaviors, and track the user's past trajectory (e.g., where the user has been to the past) or even the real-time location. We have implemented BAT using a BLE sniffer, a customized smartphone and a Bluetooth development board, and tested our *passive* and *active* BAT attacks with 54 of our own devices and also 4 development boards, and identified 39 devices (11 centrals, 24 peripherals, 4 development boards) that use the allowlist, none of which defeats our BAT attacks currently.

We advocate the use of an interval unpredictable, central and peripheral synchronized RPA generation scheme to counter our passive BAT attacks (§6.1). To defend against our active BAT attacks, we propose adding a sequence number (which could be a timestamp) when generating the RPA to ensure that each MAC address can only be used once to prevent the replay attack (§6.2). We have developed a prototype of our defense named Securing Address for BLE (SABLE) atop Android Open Source Project (AOSP) [10] and the SDK of Nordic [11] by modifying the Bluetooth protocol stack. We have tested SABLE, and our experiment result shows that (1) our passive-attack defense will have about 1% power consumption overhead for centrals and 6.75% for peripherals, and 88.49 μs performance overhead for centrals and 94.46 μs for peripherals; and (2) our active-attack defense will not impose any power consumption for centrals but will have 3.04% power overhead for peripherals, and 63.58 μs and 20.54 μs performance overhead for centrals and peripherals, respectively.

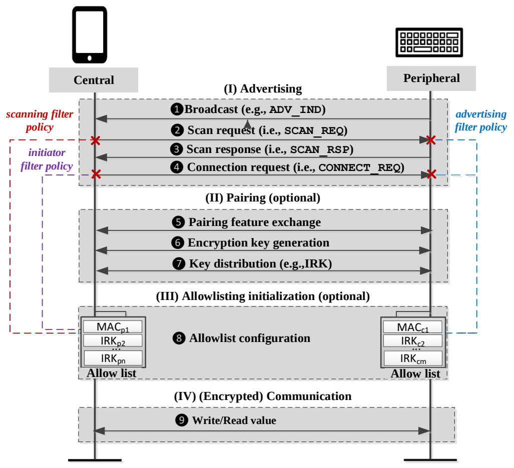
Central
Peripheral
(I) Advertising
scanning filter policy
advertising filter policy
initiator filter policy
❶ Broadcast (e.g., ADV_IND)
❷ Scan request (i.e., SCAN_REQ)
❸ Scan response (i.e., SCAN_RSP)
❹ Connection request (i.e., CONNECT_REQ)
(II) Pairing (optional)
❺ Pairing feature exchange
❻ Encryption key generation
❼ Key distribution (e.g.,IRK)
(III) Allowlisting initialization (optional)
MAC_p1
IRK_p2
IRK_pn
Allow list
❽ Allowlist configuration
MAC_c1
IRK_c2
IRK_cm
Allow list
(IV) (Encrypted) Communication
❾ Write/Read value

Figure 1: BLE workflow with corresponding allowlist policies.

2 BACKGROUND

When using BLE for communications between a central (e.g., a smartphone) and a peripheral (e.g., a keyboard), it usually involves a number of steps. As illustrated in Figure 1, it could have up to nine steps, and these steps can be categorized into four stages [6]: (I) Advertising stage, (II) Pairing stage, (III) Allowlisting initialization stage, and (IV) Data exchange stage. The details of these stages are described as follows:

(I) Advertising Stage. In this stage, the central and the peripheral establish the connection by first broadcasting the presence from the peripheral, followed by a scan request from the central, then a scan response from the peripheral, and finally a connection request from the central.

- Step ❶ Broadcast. In Bluetooth communication, the presence of a peripheral must be known to the nearby centrals. This is achieved by broadcasting the packet that includes the MAC address of the peripheral, the PDU type of the advertisement (e.g., ADV_IND which indicates that this device can be connected and scanned, or ADV_DIRECT_IND which indicates this device can only be connected by devices with the expected MAC address specified in the ADD_R field in the broadcast packet), and other optional information such as service UUIDs and manufacture data. Note that there is a special type of Bluetooth device, namely beacons, which only broadcast ADV_NONCONN_IND packets.
- Step ❷ Scan Request. When a central receives an ADV_IND packet, typically it will respond with a Scan Request (i.e., PDU type SCAN_REQ) [12]. However, since Bluetooth 4.0 (a.k.a., Bluetooth Low Energy), a new allowlist feature called *scanning filter policy* is introduced, with which a device such as a central can configure to only respond its SCAN_REQ to the allowlisted devices. This allowlist feature not only saves energy for the central, but also makes the communication more secure by only allowing connections with the listed devices. However, in practice, smartphones typically do not configure this policy as it will prevent the smartphones from discovering new peripherals.

- Step ❸ Scan Response. When receiving a `SCAN_REQ` from a central, the peripheral typically will respond with a Scan Response [8] (i.e., PDU type `SCAN_RSP`). However, similar to the central, which can have a *scanning filter policy*, a peripheral can have an *advertising filter policy*, allowing the peripheral to only respond its `SCAN_REQ` to its allowlisted device. In addition to all the advantages of using allowlist in centrals, using allowlist in peripherals also enables security and privacy protection. In particular, Bluetooth protocol specification recommends advertising sensitive data (e.g., static data such as manufacturer information, device type such as keyboard) in `SCAN_RSP` (not in `ADV_IND`) [9] to only trusted devices (i.e., the ones in its allowlist).
- Step ❹ Connection Request. When the central receives a `SCAN_RSP`, it then determines if the peripheral is of its interest (e.g., a keyboard that is of interest to its OS, or a blood pressure that is of interest to its corresponding app). By default, the OS of the central will not automatically initiate a Connection Request (i.e., PDU type `CONNECT_REQ`) to the peripherals, a user or an app has to be involved to initiate the connection. However, after the central has paired with the peripheral, it can maintain the paired peripheral into its allowlist with an *initiator filter policy* to decide whether to automatically (much like a magnet) initiate the connection whenever it sees the corresponding peripheral. The use of this allowlist policy can significantly improve the user experience, since it does not require the user to manually open the `settings` app of the OS or other 3rd-party apps to initiate the connection. For the peripheral, when receiving a `CONNECT_REQ`, it can also use the *advertising filter policy* as in Step ❷ to decide whether to accept this `CONNECT_REQ` in case that a central directly connects to it without using `SCAN_REQ` [9].

(II) Pairing Stage. Pairing, which is optional, is used to negotiate cryptographic keys for communication security and privacy and can be broken into three steps (from Step ❺ to ❼). In particular, Step ❺ Pairing Feature Exchange: the two devices exchange their pairing features (e.g., having a display, or a keypad), which are needed to decide the appropriate pairing method such as Just Works, Passkey Entry, Out of Band, and Numeric Comparison (since Bluetooth 4.2). Step ❻ Encryption Key Generation: the two devices determine the type of pairing method based on the features exchanged and negotiate an encryption key. Step ❼ Key Distribution: two devices exchange keys, and these include the encryption key and also the Identity Resolving Key (IRK), which is used for a BLE device to resolve its peer's randomized MAC address.

MAC address randomization is crucial for BLE security and privacy [13][14]. A Bluetooth MAC address is a 48-bit value uniquely associated with the device. There are four types of MAC addresses: Public Address (PA), Static Random Address (SRA), Resolvable Private Address (RPA), and Non-Resolvable Private Address (NRPA). These address types can be identified by parsing both `TxAdd` and the two most significant (`MSB`) bits of the randomized MAC address.

- **Public Address (PA)** (`TxAdd=0`): a PA is a globally static address assigned by the manufacturer. It never gets changed (serving as an identity for the device), obviously vulnerable to MAC address tracking attacks.
- **Static Random Address (SRA)** (`TxAdd=1`, `MSB=11`): an SRA is randomly generated by the device when it is rebooted or reset [6]. SRA is also vulnerable to MAC address tracking if the device never reboots or resets.
- **Resolvable Private Address (RPA)** (`TxAdd=1`, `MSB=01`): an RPA is generated using an IRK and it changes periodically. Only the paired device with a valid IRK can resolve the corresponding RPA to identify the known devices.
- **Non-Resolvable Private Address (NRPA)** (`TxAdd=1`, `MSB= 00`): an NRPA is randomly generated and changes periodically depending on the implementation. NRPA is intended to never be resolvable by any device.

It can be noticed that only RPA can still be resolved by the peer devices if they know the corresponding IRK. This is particularly useful for a peripheral to remember the recognized centrals or vice versa. Next, we explain the format of RPA and its generation and resolution process. In particular, according to the Bluetooth protocol specification [9], an RPA consists of prand and hash. The MSB of prand for RPA is fixed (i.e., `MSB=01`), and the rest of the prand are the random bits.

- **RPA Generation**. To generate an RPA (48-bits), the central, denoted with symbol c, first selects a 24-bits prand (i.e., prand_{24} whose first two bits are predefined), and then it feeds its IRK, assume irk_c, along with the selected prand into a pre-defined hash function H to get a 24-bits hash value $H_{24}(\text{prand}_{24} \mid\mid irk_c)$. Finally, the RPA of c, assume rpa_c, is generated by concatenating prand and the hash value:

$$rpa_c = \text{prand}_{24} \mid\mid H_{24}(\text{prand}_{24} \mid\mid irk_c)$$

- **RPA Resolution**. When receiving an rpa_c from a central, the peripheral can determine whether this RPA is from its "known" device. This is achieved through the RPA resolution. At a high level, the peripheral will first split rpa_c into two parts: prand_{24} and hash_{24}. Next, it iterates its known IRK list (each element of this list is added during the pairing), assume irk_i, to compute

$$\text{hash}'_{24} = H_{24}(\text{prand}_{24} \mid\mid irk_i)$$

If hash'_{24} matches the received hash_{24} split from the rpa_c, then the device is resolved with the corresponding irk_i.

(III) Allowlisting Initialization (Step ❽). This is also an optional stage depending on the configurations, and it is used to configure the allowlist used by early Steps (e.g., ❷, ❸) for device filtering. To uniquely identify a device, the allowlist feature relies on the IRKs transmitted at Step ❼, and Step ❽ just adds the IRKs to the list with other information, such as the address type. However, if the added device does not enable the address randomization, then the MAC address of the device instead of its IRK will be added to the allowlist.

Recall that there are three allowlist-based filtering polices: (*i*) *scanning filter policy*, (*ii*) *advertising filter policy*, and (*iii*) *initiator filter policy*. Among them, the *advertising filter policy* is deployed at the peripheral side, and the other two are deployed at the central side. Although the Bluetooth protocol does not specify how many devices can be added to an allowlist, we find that different policies may affect the number of devices that can be added to the allow list. In the following, we summarize their practical impacts when used:

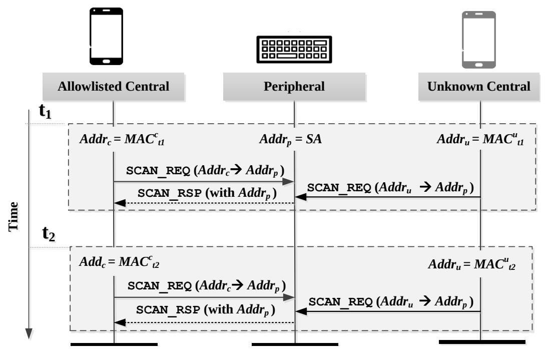

Figure 2: An example to demonstrate the key observation.

- **Scanning filter policy.** While the *scanning filter policy* is defined in the specification, it is not appropriate to be deployed on smartphones, since it will prevent smartphones from discovering new BLE devices. Instead, it is usually deployed in some customized controllers (e.g., wireless controller for video games [15]), where one controller only discovers and communicates with only one BLE device.
- **Initiator filter policy.** This policy is widely deployed in smartphones for auto-connection without user involvement if a known peripheral is detected within its reach. Since it will not prevent smartphones from discovering and connecting new BLE devices, a smartphone with this policy enabled will be able to automatically connect multiple peripherals for user convenience.
- **Advertising filter policy.** We find that in practice only one allowlisted central can be added when enabled the *advertising filter policy* in a peripheral. This is because once the peripheral has paired with a particular central, it usually restricts other centrals to connect and pair with it. Without going through the pairing process, other centrals will not be able to deliver their IRKs to it and the allowlist will not be able to add any new centrals. As such, this policy usually allows a consistent one-to-one mapping between the allowlisting peripheral and the listed central.

(IV) Data Exchange Stage (Step ❾). After the first three stages, the two devices can now exchange data using a client-server (C/S) mode, either using encryption if they have exchanged cryptographic keys or plaintext if not. Specifically, the central plays the client role, and the peripheral acts as a server providing services to the client. A read request can be sent to the peripheral if the central needs to read data from the server, or with a write request if the central needs to submit data to the server.

3 OVERVIEW OF BAT ATTACKS

3.1 Key Observation

A key observation of our BAT attacks is that once a device has been configured with allowlisting, it will behave differently when responding to its allowlisted devices and non-allowlisted devices, as described in §2. For example, if a peripheral enables *advertising filter policy* to filter the SCAN_REQ from the nearby centrals, it will always respond to its allowlisted devices while ignoring the SCAN_REQ sent from the non-allowlisted devices. Specifically, as illustrated in Figure 2, at time t_1, assume there is a peripheral p with static address SA (for simplicity, assume p uses a static address at this moment), which is broadcasting, and there are multiple centrals nearby (assume an allowlisted central c and an unknown central u). We can observe that p only accepts the SCAN_REQ from c with $MAC^c_{t_1}$ with a SCAN_RSP, but ignores the SCAN_REQ from the central u with address $MAC^u_{t_1}$. As such, it is quite straightforward to identify and track the allowlisted central c, by simply associating the randomized MAC addresses together from the sniffed packets. This is because the *advertising filter policy* creates a consistent one-to-one mapping between the allowlisting peripheral p and the allowlisted central c, enabling the attacker to uniquely associate the central of interest.

3.2 Objectives, Assumptions, and Attack Model

Attack Objectives. We consider the users whose Bluetooth devices use RPA as our victims, since a Bluetooth device can be easily tracked if its MAC address is not randomized. Today, many Bluetooth devices have adopted randomized MAC addresses such as RPAs. For instance, Google has enforced the use of RPA on all Android smartphones since 2016 [16]. The goal of BAT attack is to show that the Bluetooth devices with RPA can still be tracked, allowing their users to be potentially deanoymized (e.g., when the MAC address can be associated to a particular user).

Without loss of generality, we define the objective of BAT attacks as follows: for a set of sniffed MAC addresses (regardless of how many BLE devices they belong to), assume that

$$G(MAC) = \{MAC^{dev_1}_{t_1}, MAC^{dev_2}_{t_2}, ..., MAC^{dev_n}_{t_n}\}$$

where $MAC^{dev_x}_{t_x}$ is the MAC address of device dev_x at the time t_x. For any two MAC addresses, assume that

$$MAC^{dev_a}_{t_a} \in G(MAC), MAC^{dev_b}_{t_b} \in G(MAC), |t_a - t_b| > T$$

where T is the randomization time interval (e.g., 15 minutes), the goal of BAT attack is to determine whether $dev_a = dev_b$. If so, the attacker successfully associates the two MAC addresses. For example, assume at time t_a the attacker observed a victim is at her office and sniffed an address $MAC^{dev_a}_{t_a}$, and at time t_b, the attacker sniffed an anonymous device with $MAC^{dev_b}_{t_b}$ in a Starbucks. If the attacker identifies that $MAC^{dev_a}_{t_a}$ and $MAC^{dev_b}_{t_b}$ are the ephemeral addresses of the same device (e.g., a user's smartphone), the attacker then successfully associates these two addresses, and knows the user is in or near the Starbucks.

Victim Settings. Not all users whose BLE device configured with RPA can be attacked, and instead our BAT attacks have the following two requirements:

- We require the victim to have a paired central and peripheral, and either the central or the peripheral has configured the allowlist. This is true for most centrals and peripherals today, since centrals often configure the *initiator filter* for automatic connection with known peripherals, and many peripherals often configure the *advertising filter* to respond only to known centrals.
- We require that the central and the peripheral are not always connected, which is also often true. For example, a smart speaker is disconnected from the mobile when the user is away, and earbuds are disconnected from the mobile when they are in their

charging cases. If they are connected, attackers have to wait until either the two devices begin to communicate, or one of them is available for attackers to communicate. Although the attacker can use jamming to intentionally disconnect devices, users may notice that two connected devices are disconnected, which will break the stealthiness of the attacks.

Attack Scenarios. Since the attacker is able to either (*i*) passively sniffing the BLE traffic or (*ii*) actively modifying the traffic, it leads to two types of BAT attacks:

- **Passive** BAT **Attacks (§4)**, which just passively sniff the advertising packets between centrals and peripherals, to see whether a peripheral will selectively respond to SCAN_REQ (i.e., reply with a SCAN_RSP or not); if so, attackers use this single bit to associate (i.e., track) the corresponding centrals into unique ones based on the observed MAC addresses.
- **Active** BAT **Attacks (§5)**, which actively manipulate the packets, e.g., forging new packets, or replaying the old packets, to observe how a peripheral or a central behaves. If there are distinctive behaviors observed, their corresponding MAC addresses are associated too.

Attacker’s Assumptions. When launching BAT attacks, we assume the attackers to have the following capabilities:

- When launching passive BAT attacks, we assume that attackers can keep collecting the advertising packets (e.g., ADV_IND, SCAN_REQ, SCAN_RSP) sent from the victim’s devices, across the randomization time interval. To this end, the attacker has to be close to the victims to sniff the Bluetooth packets, and the victim devices have to be turned on (e.g., many battery-powered peripherals may enter the sleep mode when they are not in use for a while and the attacker will have to wait until they have been awakened up).
- When launching active BAT attacks, in addition to assuming that attackers can collect Bluetooth packets, we also assume that they can actively replay and forge the advertising packets (which are not encrypted). This can be easily achieved using Bluetooth development boards or smartphones.

While we have made these assumptions, we believe that our BAT attacks are still practical compared to other Bluetooth device tracking attacks. Particularly, the first assumption is an assumption for all the Bluetooth tracking attacks (e.g., [3, 17–19]). This is because Bluetooth is designed for short range communication, and the Bluetooth packets can always be collected by either using Bluetooth sniffers at multiple locations or moving the sniffer around to track potential victims. Moreover, we can also get rid of this assumption by considering smartphones to be sniffers once controlled by malware. In particular, a malware can be installed onto smartphone to force a mobile device to work as a sniffer (e.g., as in BadBluetooth [20] and Bluetooth Misbonding [21], which assumed that the malware can be installed onto mobiles), our attacks then will not have any range limitations. However, for simplicity and practicality, we do not assume that we have a malware-controlled smartphone sniffer, though nation state can have such a capability with 0-click exploits [22]).

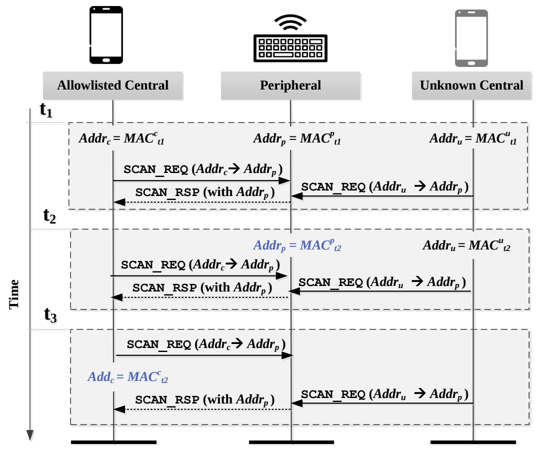

Figure 3: Workflow of our passive BAT **attacks.**

3.3 Scope

Victim Scope. While our BAT attacks can be launched to work against all the devices that satisfy the victim’s settings (discussed in §3.2), we particularly focus on the following types of victims. (i) Both the central and peripheral have enforced the randomized MAC addresses. Although in §3.1 we use the case where the central uses random MAC addresses but the peripheral uses a static MAC address to explain the key observation of our attacks, we do not consider such a vulnerable case since the Bluetooth specification has already offered the defenses for these attacks — both devices enforce the MAC address randomization. (ii) Multiple BLE devices are nearby. This is because if there is only one device (i.e., the victim) at a specific location, and if the attacker knows such a fact, he or she can easily identify the victim device without launching our attacks at all. (iii) While our attacks can track both the peripherals and the centrals, we focus on tracking victim centrals for proof-of-concept since many peripherals (e.g., keyboards and mouses) are stationary.

Attack Scope. We also would like to narrow down the scope of our attacks. (i) We only study the BAT attacks in the advertising stage (where devices may exchange ADV_IND, SCAN_REQ, SCAN_RSP and CONNECT_REQ), and we do not consider the tracking attacks in the other three stages, as the allowlist only takes effects at this stage. For instance, we do not consider attacks such as BIAS [23], KNOB [24], and Downgrade [25] (those attacks are usually launched in the communication stage or the pairing stage), although they can potentially be used for device tracking. (ii) While our attacks can be targeted (e.g., the attacker knows a MAC address of a specific device) and untargeted (e.g., the attacker does not know such a MAC address), we use targeted attacks to demonstrate the impacts. Specifically, for proof-of-concept, we assume that for a particular targeted user (e.g., Alice), we know an instance of the MAC address of her BLE device (e.g., her smartphone). There are multiple ways to learn such MAC addresses (e.g., physically following Alice with a sniffer), and similar to other works that against Bluetooth privacy (e.g., [26],[27]), we consider those to be out of the scope of this work. Please also note that without knowing such a MAC address, the attacker can still launch untargeted attacks to track arbitrary users if they are in the attacker's range.

4 PASSIVE BAT ATTACKS

Our passive BAT attacks only sniff the broadcasting traffic of BLE devices and rely on the different response signals from devices with allowlist to associate (i.e., track) the sniffed MAC addresses. Note that we focus on the scenario in which both centrals and peripherals have used RPA-type of MAC addresses; otherwise, it is trivial to recognize the allowed centrals if the peripheral does not change its MAC address as illustrated in §3.1. Theoretically, when both peripheral and central have used RPA perfectly, the BAT attack, which relies on the different response signals, will fail. However, we notice that, although both the central and the peripheral will change their addresses, the address randomization usually occurs independently (without any synchronization), and there is often an interval, where one of the devices changes its address and the other does not, leaving footprints for an attacker to track Bluetooth devices across randomization time interval. To be more specific, the root causes of the vulnerability for such passive attacks are:

- **Predictable Randomization Time Interval.** The Bluetooth devices usually use predictable intervals (e.g., 15 minutes). As such, attackers can use timing side channel to easily link a specific device by observing that the device changes its MAC address with fixed time intervals. For example, if there is one smartphone and it always changes addresses every T_r minutes, a passive attacker can observe there is one device re-randomizes its addresses at time t_1, $t_1 + T_r$, $t_1 + 2 * T_r$, respectively. Then, it would be trivial for the attacker to link the device though it has changed its MAC.
- **Asynchronized Randomization.** Even if both the central and the peripheral use unpredictable randomization time intervals (e.g., both devices change MAC address randomization time interval independently), there is no guarantee that the two devices will simultaneously change their addresses. For example, assume that there are two devices, and both of them change their MAC addresses every 15 minutes. For the first time, the central changes its MAC address at t_0, but the peripheral changes its MAC address at t_1 ($t_0 \neq t_1$). As such, there is the interval $t_1 - t_0$, where one device changes its address and the other remains the same.

We now explain the attacks in greater detail. Since broadcasting traffic starts from peripherals, we have to first identify the allowlisting peripheral from the sniffed BLE packets, and then associate the randomized sniffed MAC addresses to the corresponding centrals. To be more specific:

- **Allowlisting Peripheral Identification.** To identify whether a peripheral enables allowlist or not, we can just observe how a SCAN_REQ gets responded (i.e., whether the peripheral only responds to the SCAN_REQ sent from a specific central). Although SCAN_RSP only contains the source of the response (i.e., it contains the MAC address of the peripheral but not the MAC address of the central) and we cannot know the destination of the response, we can observe which SCAN_REQ (which contains both central's MAC address and the peripheral's MAC address) triggers the SCAN_RSP. As shown in Figure 3, if we observe (i) a peripheral always ignores the SCAN_REQ containing MAC addresses of other centrals (e.g., $MAC^u_{t_1}$), the peripheral is an allowlisting peripheral; (ii) the allowlisting peripheral receives a SCAN_REQ containing the MAC address of a specific central (e.g., $MAC^c_{t_1}$), the peripheral sends a SCAN_RSP, we know that the MAC address belongs to the allowlisted central.
- **Address Association.** Since the central and the peripheral will not change their addresses at the same time, the attacker can always observe an interval, during which one device does not change its address, and such a pattern can be used to track its peer. For example, in Figure 3, assume that the peripheral first changes its address to $MAC^p_{t_2}$, while the central MAC address remains $MAC^c_{t_1}$ during the small interval $T = t_3 - t_2$. Then, by observing that the peripheral with $MAC^p_{t_2}$ only responds to the central with $MAC^c_{t_1}$ and ignores the SCAN_REQ from others, we know that the peripheral is likely to be the same peripheral (or another peripheral whose allowlisted central is also the same central). Later, at t_3, when the central changes its address, the peripheral remains $MAC^p_{t_2}$. Again, by observing the central to which the peripheral with $MAC^p_{t_2}$ always responds, we conclude that the central is the same central (given the peripherals can only add one allowed central, as discussed in §2). Note that since the attack workflow for the case where the central first changes its MAC address is similar to the case described above, we omit its details for brevity.

Attack Example I: Monitoring a Victim's Behavior. The passive BAT attack is particularly useful to monitor the user's behaviors in a specific location (e.g., user's house), which may breach user's security and privacy. Assuming victim Alice is using a stationary allowlist-enabled Bluetooth-keyboard in her home to connect her smartphone, a passive attacker is able to sniff the exchanged packets (up to 2,000-ft away when using an amplified antenna such as RP-SMA-R/A [28]) between the keyboard and the smartphone, so that the attacker is able to know the latest address of the Alice's smartphone via passive attacks. Then, whenever the smartphone communicates with any peripherals, the attacker is able to associate the peripherals to Alice and infer Alice's privacy-sensitive behavior. For example, the attacker could infer that Alice may be plagued by hyperglycemia or other blood glucose disorders if the attacker observes that her smartphone communicates with a Bluetooth blood glucose monitor (the blood glucose monitor can be recognized from the services UUID [19] provided by the Bluetooth device).

5 ACTIVE BAT ATTACKS

As discussed, the passive attack requires the attacker to keep monitoring the traffic between the two devices, and it will fail when the central and the peripheral are far away. Therefore, we have to look for other techniques, and this leads to the design of our active BAT attack, with which attackers can actively inject (e.g., via forging) traffic to BLE devices to observe how they will respond, and this practice does not require the attacker to keep monitoring the traffic. This is possible, since it is extremely easy for an attacker to program a malicious central (e.g., a smartphone) or a malicious peripheral (e.g., a development board) to broadcast arbitrary packets of interests. Therefore, the key question becomes what kind of packets the attacker has to forge.

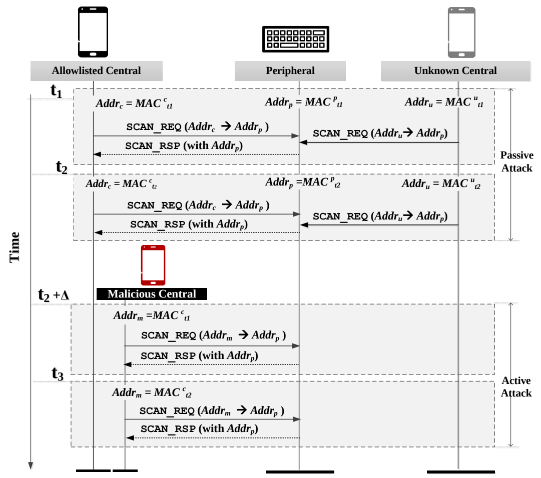

Figure 4: Workflow of BAT **attacks from malicious centrals.**

When inspecting the BLE traffic of the advertising stage, as shown in Figure 1, we can notice that there is no encryption, and thus the attacker can inject arbitrary packets. The only secret is the randomized MAC addresses sent by centrals and peripherals. However, when devices using allowlist, they must use RPA that relies on the exchanged IRKs for their randomized MAC addresses generation. Therefore, if we can replay the sniffed RPA-type MAC addresses to probe how a central or a peripheral would respond, then the attacker can still launch the BAT attack.

Surprisingly, we find that current RPA generation algorithm unfortunately never considers replay attacks, and our active BAT attack can indeed work. In particular, based on the RPA generation and resolution process described in §2, we find that the current Bluetooth specification only requires that a random number to be used in the RPA generation, but it does not specify how to choose this random number (e.g., whether the device can choose a previously used random number). As a result, a randomized MAC address generated from a previously used random number is also considered valid, even though the attacker does not know the corresponding IRK. Therefore, a malicious central or peripheral can create a spoofed packet with a previously used MAC address to probe the peripheral or central. Since the replay attack can be launched from either a malicious central (§5.1) or a malicious peripheral (§5.2), we describe these two types of active BAT attacks, respectively.

5.1 Active BAT from Malicious Centrals

The goal of a malicious central based active BAT attack is to use a malicious central to first identify the allowlisting peripheral, from which to further track the allowlisted central. As illustrated in Figure 4, in addition to using the same approach to identify the allowlisting peripheral as in the passive BAT attacks (described in §4), it only adds one additional step of actively injecting a SCAN_REQ at $t_2 + \Delta$ with the harvested $MAC^c_{t_1}$ at t_1, where $|t_2 - t_1| > T$, to probe whether the MAC address randomized peripheral responds to this injected address. If so, the peripheral is identified first, from which attackers can then track the potential centrals by actively injecting the collected the MAC addresses of these centrals.

More specifically, as illustrated in Figure 4, at $t_2 + \Delta$, an attacker can use a malicious central to create a SCAN_REQ with the previously used $MAC^c_{t_1}$ (which was sniffed somewhere), and then sends this request to *all* allowlisting peripherals. If one of them (e.g., the peripheral with a $MAC^p_{t_2}$) responds with a SCAN_RSP, the attacker identifies that this peripheral p is the one paired with the victim central (having the central's IRK and recognizing it). At time t_3, the attacker can further associate the address of $MAC^c_{t_2}$ to $MAC^c_{t_1}$, since the attacker knows that the identified p is supposed to only respond to this allowlisted central c. The active attacks from malicious centrals are more powerful than passive attacks, and they can be used to track both the (past) trajectory and real-time locations of a victim. In the following, we provide two examples to demonstrate how these can be achieved.

Attack Example II: Tracking a Victim's Past Trajectory. Assume that Alice is using her smartphone to communicate with her stationary allowlist-configured Bluetooth keyboard p in her workspace, and the attacker is able to collect one of its MAC addresses (assume $MAC^p_{t_1}$). When at t_2 Alice is away from her workspace, the attacker aims to know where Alice has been to. To this end, the attacker deploys beacon-alike sniffers (everywhere or just a few targeted places) that broadcast the ADV_IND packets with $MAC^p_{t_1}$ to collect the MAC addresses of all nearby centrals in their SCAN_REQs (assume $MAC^{c_i}_{t_2}$, $MAC^{c_j}_{t_2}$, $MAC^{c_k}_{t_2}$). Then, at t_3, the attacker moves closer to Alice's workspace, and uses a malicious central to replay the collected SCAN_REQs with $MAC^{c_i}_{t_2}$ to Alice's p to test whether it responds. If so, the attacker knows c_i is Alice's phone, and if $MAC^{c_i}_{t_2}$ was collected from Starbucks (based on the sniffer's location), the attacker knows Alice was in (or near) the Starbucks.

Attack Example III: Tracking a Victim's Real-time Location w/ Tunneling. In our attack example II, the attacker has to wait at t_3 to detect Alice's past trajectory, because there is no direct communication channel between the wild centrals and p. Therefore, if the attacker is able to build a tunnel to relay the sniffed SCAN_REQs directly to p in Alices' workspace, then he or she would be able to know Alice's location instantly. This leads to our 3rd attack example, which is to additionally build a tunnel between the wild sniffers and p using attacks such as the wormhole attack [29]. Details are omitted since such a tunneling attack is well-known, and also the rest is similar to attack example II.

5.2 Active BAT from Malicious Peripherals

Using malicious centrals to probe true peripherals relies on the allowlist of the *advertising filter policy* in the peripherals. However, not all peripherals enable this policy, and rather many centrals (e.g., Android mobiles, iPhone and Windows tablets) have enabled the *initiator filter policy*, which will instantly respond to the "known" peripherals (by storing the peripheral's IRK) once they are in their range. Therefore, we design another active attack by using spoofed packets generated from a malicious peripheral, which broadcasts the advertising packets to all nearby centrals, and only the central enabled the *initiator filter policy* will respond, allowing an attacker to instantly know a central's location. Similar to active BAT attacks

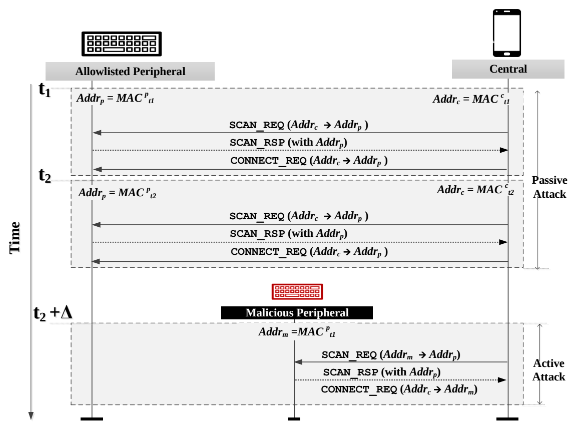

Figure 5: Workflow of BAT **attacks from malicious peripherals**

from malicious central, this attack also does not need uninterrupted observation of Bluetooth traffic.

More specifically, as illustrated in Figure 5, assume an attacker has observed that a central with $MAC^c_{t_1}$ initiated a connection with a peripheral with $MAC^p_{t_1}$ at t_1. At t_2, the central changed its address to $MAC^c_{t_2}$ and the peripheral changed its address to $MAC^p_{t_2}$. To associate whether the address $MAC^c_{t_1}$ and $MAC^c_{t_2}$ belong to the same central, at $t_2 + \Delta$, the attacker can simply create a malicious peripheral to broadcast its presence with the harvested $MAC^p_{t_1}$ or $MAC^p_{t_2}$. If the victim central has enabled the *initiator filter policy*, it will automatically initiate `CONNECT_REQ` to the malicious peripheral. Consequently, the attacker is able to associate $MAC^c_{t_1}$ with $MAC^c_{t_2}$ and identify the targeted central.

Attack Example IV: Tracking a Victim's Real-time Location w/o Tunneling. Building a tunnel might be expensive. There is no need to do so if attackers use malicious peripherals to associate centrals with the *initiator filter policy*. Still assume that Alice uses her phone configured the *initiator filter policy* to automatically communicate with her Bluetooth keyboard p in her workspace, and at t_1 the attacker is able to observe one of its MAC addresses $MAC^p_{t_1}$. Then later at t_2 when Alice is away from her workspace, the attacker directly uses beacon-alike sniffers to advertise $MAC^p_{t_1}$ to nearby centrals, and if a central instantly connects the sniffer with a `CONNECT_REQ`, then the attacker knows Alice's real-time location.

6 COUNTERMEASURE

In this section, we present a countermeasure named **S**ecuring **A**ddress for **BLE** (SABLE) to defend against our BAT attacks. Since the attacks can be launched passively or actively, we need the following two corresponding defenses described in §6.1 and §6.2, respectively.

6.1 Defending Against Passive BAT Attacks

Overview. The passive attacks are made possible due to the central and peripheral independently randomizing their addresses. As such, to defend against the passive attacks, we propose to (1) make MAC randomization at both sides synchronized, and (2) make the length of the interval random; otherwise, a repeatable time interval (e.g., every 15 minutes) allows attackers to associate the randomized MAC addresses across intervals.

(I) Making Randomization Synchronized. We first discuss how the two devices change their addresses and when it is the time for them to perform the address randomization. Since the central and the peripheral may not always be close to each other (e.g., the user could take her central away, and vice versa, or one of them is turned off), and if they are not close to each other, the two devices cannot communicate to decide how they could change their addresses. Therefore, we have the following two scenarios to address:

- **(a) Two devices are close to each other.** In this case, we cannot let the peripheral and the central independently start their own randomization. Instead, we notice that the allowlisting always starts from the peripheral when advertising its presence, and then the central responds. Therefore, we can take advantage of this causality for the synchronization and let the randomization start right before the peripheral starts to send the `ADV_IND` at the peripheral side, and the central starts right after receiving the corresponding `ADV_IND`, as illustrated in Figure 6. As such, attackers will only observe an always synchronized randomization across an unpredictable interval, and they cannot associate the MAC addresses across the randomization time intervals anymore.
- **(b) Two devices are far away.** In this case, we can let the central and peripheral independently start their own randomization, since the attacker can no longer associate the addresses through observing devices' SCAN_REQ and SCAN_RSP. Assume a central and a peripheral have passed N times synchronized address randomization, and now they are in their $(N+1)$-th synchronized address randomization. During this interval (e.g., 15 minutes), the central is taken away by the user, and they are no longer close to each other (we use the case where the central is away from the peripheral as an example, since the other case is essentially the same). As a result, the peripheral will not be able to receive any SCAN_REQ from its allowed central. When it is the time to change its address, the peripheral will fetch its own time T_p and generate a random address rpa_p. Since the central is not nearby, the lifetime of such a generated rpa_p can be a random time period without notifying the central for synchronization. The peripheral will continue to generate new RPAs using its freshest timestamps, and each of those RPAs will have a random lifespan. When the central is back, the peripheral will resume its state by re-entering its $(N + 2)$-th address randomization. Specifically, since the peripheral keeps broadcasting, and whenever the central is close to it, the central can recognize it by correctly resolving its RPA. Then, the central sends a SCAN_REQ, which contains its latest RPA generated from the freshest timestamp, to inform the peripheral that it is back. When the peripheral receives the RPA, it enters the $(N + 2)$-th synchronized address randomization, and then the central also enters the $(N + 2)$-th synchronized address randomization when it receives the `ADV_IND` from the peripheral. The following procedures are the same as that in scenario (a), which we will not present in further detail for brevity.

(II) Making Randomization Time Interval Consistent and Unpredictable. Our randomization time interval needs to be a secret

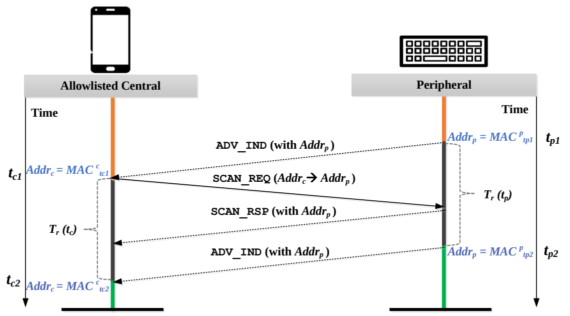

Figure 6: Status changing in the passive defense.

and unpredicatable to attackers. To achieve that, the two devices can introduce a new secret value or reuse existing ones (e.g., IRK) to derive the randomized intervals. The secret also needs to be updated dynamically on both sides to ensure that the randomization time intervals can also be changed simultaneously. As such, we propose to use the hash of an auto-incremented secret to derive the randomization time interval, since every time the value of such derived secret changes, the derived randomization time interval changes as well.

The devices can use an existing secret S (e.g., the LTK, or the IRK) with an automatic increment to derive the random time intervals. Assume there is an S_0, which can be the IRK, and every time we update the interval S_j, we automatically increase it by k (which can be one or any other predefined number) from previous S_i.

$$S_j = S_i + k$$

where S_0 is assumed to be the IRK for simplicity. Then, we can derive the length of the randomization time interval at the i-th interval

$$T_r(i) = H(S_i) \bmod T_{max}$$

where T_{max} is set to be 15 minutes, and $H(S_i)$ is the hash of S_i. As such, at any given time interval i, both central and peripheral will have a pre-determined random interval $T_r(i)$ unknown to attackers once S_0, the initial secret, is exchanged.

Also, note that there has to be a unique secret S for each paired central and peripheral. But one central (e.g., a smartphone) can be paired with multiple peripherals. Therefore, the randomization time interval and IRK must be S-specific. That is, a central needs to use a peripheral-specific RPA-type MAC address correspondingly to connect its paired peripherals. Finally, two paired devices may be out of synchronization, e.g., it is supposed to be i-th interval, but for some reasons (e.g., one of them lost its battery) the peripheral or the central may still be in the previous (i-1)-th interval or even more. If this occurs, the devices have to correct the errors based on the observed intervals. In particular, when a central notices that the two devices get out of synchronization by observing whether the peripheral's interval equals to its own currently used interval, the central starts the error handling process: the central first calculates a few intervals (e.g., (i-1)-th interval) that are close to the currently used interval (assume it is the i-th interval) based on the algorithm of how $T_r(i)$ is derived, e.g., $T_r(i-1) = H(S_{i-1}) \bmod T_{max}$, $T_r(i-2) = H(S_{i-2}) \bmod T_{max}$, and compares those calculated intervals to check if any of them equals to the peripheral's interval. If so, the central updates its i (the current interval) accordingly, and the error is corrected (the central and the peripheral now have the same i).

6.2 Defending Against Active BAT Attacks

Overview. Fundamentally, our active BAT attack works because the current RPA-type MAC address generation suffers from the replay attack. There are multiple countermeasures to defeat a replay attack, e.g., using a sequence number, or a timestamp, or storage-based (i.e., saving all the used random numbers onto the devices). However, not all of them are practical. For example, the storage-based solution may cost huge amount of storage resource, e.g., up to $365*24*4*48 = 1,681,920$ MB yearly assume address is changed every 15 minutes (note that each address is 48 bits).

Another well-known defense against replay attack is to add random sequence numbers. Theoretically, we can add a synchronized, auto-incremented sequence number, together with the random number prand, to generate a one-time only RPA-type MAC address. However, this defense will introduce additional storage and communication overhead, as well as additional sequence number maintenance efforts. For example, the central and the peripheral must update their negotiated sequence number simultaneously to ensure they keep the same sequence number. We therefore propose to use the timestamps together with the random number prand to generate a one-time only RPA-type MAC address, which can only be used within a given time window depending on the configuration. Note that a timestamp essentially can be considered as a sequence number, and it increases automatically as time passes by.

However, we did not use the timestamp directly since this will result in the collision of the Bluetooth MAC address. Recall that the Bluetooth MAC address is 48 bits, and if we include a 24-bit timestamp in the MAC address, we only have 24 bits to ensure the randomness of the MAC addresses. This can only result in 2^{24} (16,777,216) unique MAC addresses, which is much less than the current number of Bluetooth devices (e.g., annual Bluetooth device shipments is 4.7 billion in 2021 [30]). Nevertheless, given that a specific area may not have too many devices, using the timestamps directly could still be a viable mitigation approach.

Detailed Design. Our active defense is designed by piggybacking the existing protocol without adding any extra field in the protocol but only modifying the central and peripheral to process the timestamp. In total, there are 4 steps involved in our new design. Note that in the following description, we assume the peripheral has enabled allowlist (since the defense workflow for the central with allowlist is similar, we will not present it for brevity). Our defense also works when both peripheral and central enable the allowlist.

- **Step ① Key Distribution**. Assume that a peripheral p and a central c have passed through the advertising stage, and now they are entering the pairing stage. When the central distributes its IRK irk_c, it also distributes the current timestamp T_c^0 of the central (i.e., T_c^0=`getCurrentTime(`c`)` where `getCurrentTime` returns the number of seconds since UNIX Epoc for c).
- **Step ② Allowlisting Configuration**. The peripheral p receives irk_c, T_c^0, and gets its current time T_p^0 via `getCurrentTime(`p`)` at p. Then, it saves both T_p^0 and the received irk_c, and T_c^0 for future reference when resolving the RPA. Now p finishes configuring its allowlist, with which p will only respond to `SCAN_REQ` or `CONNECT_REQ` from this paired and allowlisted c.

- **Step ③ RPA Generation**. Later, when the central c receives an advertising packet from p, it uses its irk_c, a random number prand_{24} (with its two MSB bits to be 01 to denote RPA-type MAC address), and the current time T_c to generate its current
$$rpa_c = (\text{prand}_{24} || H_{24}(\text{prand}_{24} \ || \ T_c \ || \ irk_c))$$
After that, c can send this newly generated rpa_c as usual in its `SCAN_REQ` or `CONNECT_REQ` to p. Please note that this timestamp T_c should be used in all subsequent sessions within a random interval (e.g., 15 minutes) to avoid the central changing its addresses too often.
- **Step ④ RPA Resolution**. When p receives a `SCAN_REQ` with an rpa_c, it resolves this rpa_c using its stored irk_c as follows. First, we follow the initial RPA resolution by splitting rpa_c into two parts: the random number prand_{24} and hash_{24}. Next, we get T_c' from calculation:
$$T_c' \ = \ \texttt{getCurrentTime}(p) - (T_p^0 - T_c^0)$$
After feeding the hash function H with three inputs irk_c, prand_{24} and T_c to compute the hashed value:
$$\text{hash}_{24}\ ' \ = \ H_{24}(\text{prand}_{24} \ || \ T_c' \ || \ irk_c))$$
If hash_{24} equals to hash_{24}', this rpa_c is resolved. If not, there could be two possibilities. (i) The time on the central and the peripheral gets out of synchronization due to unknown reasons (e.g., clock skews). In that case, the peripheral may need to enumerate a few possible timestamps within a threshold (e.g., the threshold could be one if the peripheral attempts to try the former timestamp and the next timestamp) to correct the errors and update T_c if resolved. (ii) The address is a replayed one and the peripheral should reject it. In that case, the timestamp has significant differences when compared with the saved T_c' or the current time (e.g., the difference is beyond a threshold).

7 EVALUATION

We have implemented both our attacks and defenses for the evaluation. To implement our passive BAT attack, we used Adafruit LE sniffer [31] to collect BLE advertising packets (i.e., `ADV_IND`, `SCAN_REQ`, `SCAN_RSP` and `CONNECT_REQ`). To implement our active BAT attack, we customized Android 9.0 through the Android Open Source Project (AOSP) [10] to implement both the malicious central and the malicious peripheral. To implement our defense, we used Google Pixel 2 as a central, and a Nordic NRF52 development board as a peripheral for testing our SABLE. In the rest of this section, we present our evaluation results.

7.1 Experiment Setup

Due to ethics concerns, we can only launch our BAT attacks against our own devices configured with allowlist. Intuitively, keyboards, mouses, earbuds, and smart watches usually support a single user, and they tend to have an allowlist. We therefore first purchased 43 Bluetooth peripherals such as keyboards, earbuds, and mouses, and among them, 24 of them have the allowlist enabled. Other 19 peripherals, which do not have allowlist, are out of our focus. For the central devices such as smartphones, fortunately they all have the allowlist enabled, and therefore we used all of our own Bluetooth centrals (6 smartphones, 3 laptops, and 2 tablets). We also purchased a Bluetooth development board each from TI, Nordic, Silicon Labs, and Cypress, respectively. As reported in Table 1, we therefore eventually have 39 tested devices in total (11 centrals, 24 peripherals, 4 development boards).

Peripherals & Development Boards											
Brand & Model	Allowlist		Device Type	MAC Addr	Power Saving	Passive Attacks		Active Attacks			
								From Malicious Central		From Malicious Peripheral	
	Enabled by P	Used by C				TC	TP	TC	TP	TC	TP
DRACONIC	✓	✓	Keyboard	SRA	✓	✓	✓	✓	✓	✓	✓
JellyComb	✓	✓	Keyboard	SRA	✓	✓	✓	✓	✓	✓	✓
iClever	✓	✓	Keyboard	SRA	✓	✓	✓	✓	✓	✓	✓
Microsoft (V1)	✓	✓	Keyboard	SRA	✓	✓	✓	✓	✓	✓	✓
Microsoft (V2)	✓	✓	Keyboard	SRA	✓	✓	✓	✓	✓	✓	✓
byteblue	✓	✓	Keyboard	SRA	✓	✓	✓	✓	✓	✓	✓
Logitech K780	✓	✓	Keyboard	SRA	✓	✓	✓	✓	✓	✓	✓
Logitech K830	✓	✓	Keyboard	SRA	✓	✓	✓	✓	✓	✓	✓
Logitech K380	✓	✓	Keyboard	SRA	✓	✓	✓	✓	✓	✓	✓
SXWL	✓	✓	Keyboard	SRA	✓	✓	✓	✓	✓	✓	✓
SXWL	✓	✓	Mouse	SRA	✓	✓	✓	✓	✓	✓	✓
Inphic	✓	✓	Mouse	SRA	✓	✓	✓	✓	✓	✓	✓
Vogek	✓	✓	Mouse	SRA	✓	✓	✓	✓	✓	✓	✓
JellyComb (V1)	✓	✓	Mouse	SRA	✓	✓	✓	✓	✓	✓	✓
JellyComb (V2)	✓	✓	Mouse	SRA	✓	✓	✓	✓	✓	✓	✓
SEENDA	✓	✓	Mouse	SRA	✓	✓	✓	✓	✓	✓	✓
MiBand 4C	✓	✗	Wristband	PA	✗	✓	✓	✓	✓	✗	✓
i-Home Alexa	✗	✓	Speaker	PA	✓	✗	✓	✓	✓	✓	✓
TEZO	✗	✓	Earbuds	PA	✓	✗	✓	✓	✓	✓	✓
Boltune	✗	✓	Earbuds	PA	✓	✗	✓	✓	✓	✓	✓
SoundBot	✗	✓	Earbuds	PA	✓	✗	✓	✓	✓	✓	✓
Riitek	✗	✓	Keyboard	PA	✓	✗	✓	✗	✓	✓	✓
Cimetech	✗	✓	Mouse	SRA	✓	✗	✓	✗	✓	✓	✓
Ergonomic	✗	✓	Mouse	SRA	✓	✗	✓	✗	✓	✓	✓
TI CC2640R2F	✓	✓	Dev Board	RPA	-	✓	✓	✓	✓	✓	✓
Nordic NRF52	✓	✓	Dev Board	RPA	-	✓	✓	✓	✓	✓	✓
Silicon Labs 6101D	✗	✓	Dev Board	RPA	-	-	-	✗	✗	✓	✓
Crypess CY8kCIT	✗	✓	Dev Board	RPA	-	-	-	✗	✗	✓	✓

Centrals											
Brand & Model	Allowlist		Type & OS	MAC Addr	Random Interval	Passive Attacks		Active Attacks			
								From Malicious Central		From Malicious Peripheral	
	Enabled by C	Used by P				TP	TC	TP	TC	TP	TC
Google Pixel 4	✓	✓	Phone (Android 11)	RPA	5-15	✓	✓	✓	✓	✓	✓
Google Pixel 2	✓	✓	Phone (Android 10)	RPA	5-15	✓	✓	✓	✓	✓	✓
Samsung S10	✓	✓	Phone (Android 10)	RPA	5-15	✓	✓	✓	✓	✓	✓
Google Piexl 4	✓	✓	Phone (Android 10)	RPA	5-15	✓	✓	✓	✓	✓	✓
iPhone 8	✓	✓	Phone (iOS 13.2)	RPA	15	✓	✓	✓	✓	✓	✓
iPhone 11	✓	✓	Phone (iOS 13.2)	RPA	15	✓	✓	✓	✓	✓	✓
iPad	✓	✓	Tablet (iOS 13.2)	RPA	15	✓	✓	✓	✓	✓	✓
Dell GD1H4KU	✓	✓	Laptop (Windows 10)	PA	$+\infty$	✓	✓	✓	✓	✓	✓
Dell	✓	✓	Laptop (Ubuntu 20.02)	PA	$+\infty$	✓	✓	✓	✓	✓	✓
Thinkpad T450s	✓	✓	Laptop (Windows 8)	PA	$+\infty$	✓	✓	✓	✓	✓	✓
Surface Pro	✓	✓	Tablet (Windows 10)	PA	$+\infty$	✓	✓	✓	✓	✓	✓

Table 1: Evaluation results of the tested devices. "TC" means tracking the central and "TP" means tracking the peripheral. "✓" means vulnerable and "✗" means not.

7.2 Evaluation of Passive BAT Attacks

To test our attacks against our own centrals and peripherals, we first need to pair them up. To this end, for each peripheral, we iterate our centrals: if any of them can expose the vulnerability for the peripheral, we stop pairing more centrals and move to the next peripheral. Also, during our pairing, if a device never involves allowlist (including by its peer device), we discard it in our evaluation. As reported in Table 1, there are 17 peripherals configured with the allowlist, as shown in the 2nd column. Unfortunately, all of them are vulnerable to passive BAT attacks to track their corresponding centrals since they used static addresses, which are subject to tracking without deploying our attacks at all as described in §3.1. We can also notice that these peripherals include keyboards, mouses, and smart wristbands. As expected, these devices are intended for exclusive use (a single user) and tend to have the allowlist. However, other IoT devices such as smart speakers tend not to have the allowlist since they are household devices (serving more than one user).

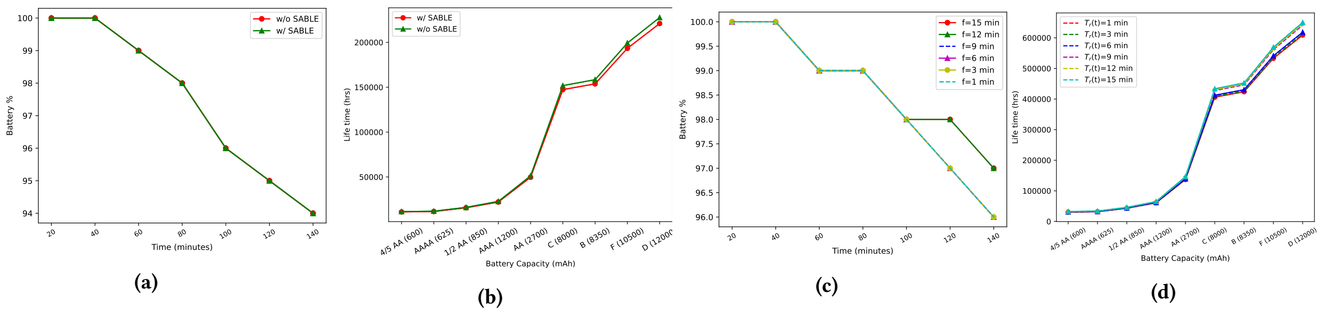

Figure 7: Power Consumption of SABLE defense.

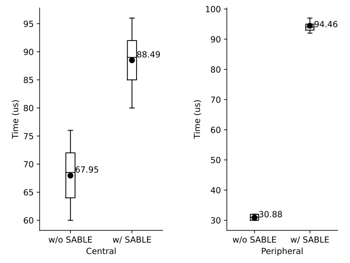

Figure 8: Performance of SABLE for active defense

Since all the peripherals are vulnerable to the simple passive attacks described in §3.1, we have not validated the passive attacks introduced in §4. To really evaluate these attacks, we use 4 development boards with RPAs, and pair the smartphone with them. Among them, only TI CC2640 and Nordic NRF52 support allowlist [32, 33], and the development boards (as well as the devices that used those development boards) are confirmed vulnerable to our passive attacks. Also, while the tested centrals are all vulnerable to BAT attacks, we notice that the randomization interval is varied from the operating systems. For example, Android-11 changes its interval to every 5 to 15 minutes, but iOS stays a constant 15 minutes. Meanwhile, Windows and Linux laptops use public addresses without randomization. In addition, as reported in the 6th column of Table 1, among the tested peripherals, we notice that only the wristband will always stay awake.

7.3 Evaluation of Active BAT Attacks

Active BAT Attacks from Malicious Centrals. Since the active BAT attacks from malicious centrals require the victim peripherals to have allowlist (regardless of whether or not they have used static addresses) and configure *advertising filter policy*, and in our passive attacks we have already identified that there are 17 peripherals configured with allowlist (as reported in the second column of Table 1), all of them can be used to launch BAT from malicious centrals to track victim's centrals. When configuring the 4 development boards with RPAs, only TI CC2640 and Nordic NRF52 can be used to track both the central and the peripheral from a malicious central, since only they support the allowlist [32, 33].

Active BAT Attacks from Malicious Peripherals. Similarly, since the active BAT attacks from malicious peripherals leverage the *initiator filter policy*, which requires the central to add the peripherals to its allowlist for auto-connection, for each central we iterate our peripherals to pair it and then test whether the central will initiate the connection requests without manually triggering the connection in the `Settings` app. If so, the peripherals can be used to launch BAT attacks from malicious peripherals to track their centrals. As reported in Table 1, all of the tested devices are subject to tracking of peripherals (the last column), and tracking of centrals except the tested Wristband since its central does not use allowlist.

7.4 Evaluation of SABLE

Effectiveness of SABLE. We launched both our passive and active attacks again against our patched Google Pixel 2 and NRF52, and confirmed that the attacks no longer work. In particular, (*i*) in passive attacks, where MAC addresses randomize on both sides in a synchronized random interval, attackers cannot observe predictable patterns between centrals and peripherals; (*ii*) in active attacks, SABLE rejected all reused packets, and attackers could not link any used RPAs to the new ones by replaying (including via tunnelling).

Power and Performance Overhead. Since our defense introduces extra operations, which will certainly result in both power consumption and performance overhead. To quantify the battery impact on central devices, we directly monitor it through the battery level. To quantify the battery impact on peripherals, we measured our tested NRF52 development board, with an Agilent 34410A Multimeter to sample the electric current of NRF52 that runs with our SABLE (and without SABLE), and calculate the average current I_{avg}, from which to approximate the battery life, using the standard $T_{bat} = C_{bat}/I_{avg}$ [34], where C_{bat} is the battery capacity.

- **Active Defense.** Since our active defense piggybacks the existing pairing messages, we measured the approximation of the battery life of our defense and compared it with the vanilla pairing protocol. According to our experiment, the remaining battery percentage decreases linearly, and there is no extra overhead observed, as shown in Figure 7a. Also, the average current of NRF52 running SABLE pairing process is 54.3 μA, and the average current when NRF52 running vanilla pairing process is 52.7 μA (an extra 3.04% overhead). This overhead is negligible for a battery lifetime across different battery capacities, as illustrated in Figure 7b. To measure the performance of our active defense, we run both our and vanilla RPA generation and resolution 100 times on both Google Pixel 2 and NRF52, respectively. Figure 8 shows these results. We can observe that the average overhead on NRF52 is around 63.58 (94.46-30.88) μs whereas the smartphone is around 20.54 (88.49-67.95) μs.

- **Passive Defense.** For passive defense with synchronized random intervals, it does not have any additional overhead if its $T_r(t)$ is set to be a fixed 15 minutes (as in iOS) since it does not have any additional operations. However, if $T_r(t)$ becomes smaller, there will be more frequent address randomizations, causing more overhead. To quantify this power consumption, we set 15 minutes as the baseline, vary $T_r(t)$ from 3 to 15 with a stride 3. The overhead on the central device is around 1%, even if we set $T_r(t)$=1 (i.e., randomizing at every minute). For the peripheral, as shown in Figure 7d, only 6.75% extra power consumption is introduced when $T_r(t)$=1, which is also negligible. For its performance, as shown in Figure 8, we can see that it takes around 88.49 μs and 94.46 μs for a mobile and development board to generate an address, respectively (with our active defense). Therefore, compared to the address randomization interval (e.g., a few minutes), such a tiny amount of time is negligible.

8 DISCUSSION

Practicality and Comparison. We have presented one passive BAT attack and two active BAT attacks. Next, we would like to compare the practicality (i.e., the cost, difficulty, and impact) of each attack. In particular, the cost is measured by the resources spent and also the efforts taken by the attackers; the difficulty is measured by the condition of the victim central and peripheral (e.g., whether the attacks require the two devices to be nearby); the impacts are the possible consequences caused by the attack. As summarized in Table 2, it can be observed that:

- **Passive Attacks (§4).** Since these attacks involve only sniffers, they are less costly. Meanwhile, the attacks do not require attackers to keep probing or building a tunnel. However, such attacks need uninterrupted monitoring of the traffic. Otherwise, the attacker will lose the target. Moreover, the attacks will fail when the central and peripheral are not within the reach.
- **Active Attacks from Malicious Centrals (§5.1).** When compared with the passive attacks, the active BAT attacks from a malicious central do not have the distance limit and do not require uninterrupted monitoring of the victim. Moreover, in addition to monitoring user behaviors, attacks can also be used to track a victim's past trajectory or instant location. However, such attacks are also subject to limitations: when launching the attacks, the attacker must probe multiple devices with SCAN_REQ or SCAN_RSP in order to identify the one of interest. Such operations can be costly when there are too many devices nearby.
- **Active Attacks from Malicious Peripherals (§5.2).** The active BAT attacks from malicious peripherals are most impactful, practical, and less costly. First, similar to the other active attacks, the attackers can launch such attacks without a distance limit and uninterrupted monitoring of the victims. Second, instead of probing each device independently, such attacks use ADV_IND, which is a broadcast message. That is, once the attacker broadcasts a single ADV_IND with an old MAC address, the victim central will expose itself instantly if it is nearby.

Security Analysis of Our Defenses. In our passive BAT defense, we rely on synchronized random time intervals at both central and peripheral to defeat cross interval MAC address tracking. Intuitively, since the defense requires synchronized random time intervals to work, and if the attacker can manipulate intervals by removing (e.g., through jamming) or injecting packets that are required for both the central and peripheral to execute the synchronization, it will disable our defense. However, while it is true that the manipulation may result in the denial of service (DoS) attack, the attacker cannot further track the central or the peripheral. Since the central and the peripheral can be far away from or close to each other (See §6.1), we then have the following two cases to analyze:

- **Two devices are far away.** In this case, the two devices independently perform their address randomization, and `SCAN_REQ` or `SCAN_RSP` sent from the two devices will not be used to perform the synchronization. As such, if the attacker removes some or all of those packets, the devices will behave as normal, no impact on the random time intervals. Meanwhile, since the attacker does not have the corresponding IRK, he or she cannot create and forge a packet with a valid MAC address. The attacker may also replay an used packet, but it is be prevented by our active defense.
- **Two devices are close to each other.** In this case, the devices perform randomization according to the exchanged `SCAN_REQ` and `SCAN_RSP`. First, removing some of those packets intentionally may result in the failure of the synchronization (e.g., it is supposed to be the i-th interval, but the peripheral or the central may still be in the previous (i-1)-th interval). However, an error correction mechanism can make the random time intervals to be re-synchronized. Second, forging or replaying the packets will not work too, the reasons of which are similar to the previous case, thereby omitted for brevity.

In an extreme case, the attacker may first remove some of the packets from the traffic, and then replay those "removed" packets, in the purpose of breaking our BAT active defense. Such attack will not work as well. Assume that at t_0, the attacker first removes all the packets sent from the peripheral, and then saves those packets. At t_1 (i.e., $t_0 + T$), the attacker sends out the collected packets to probe the central(s). The "true" central will be able to resolve it, and quickly notice those packets are old ones due to the significant differences between those packets and the current time (though those packets have never been observed before), thereby still defeating such attacks. Certainly, if the attacker obtains the IRK of the central or the peripheral, our defense will fail since he or she can forge the MAC address based on our proposed randomization algorithm. However, we consider this to be hard since the transmission of the IRK is protected by encryption.

Ethics and Responsible Disclosure. We did take ethics into the highest possible standard when launching our BAT attacks. First, we only performed our attacks against our own devices. Second, we have disclosed this vulnerability to Bluetooth SIG, Apple, Google, Microsoft, Texas Instruments (TI), and Nordic. Bluetooth SIG assigned CVE-2020-35473 to track this logical-level design flaw, Google assigned Android-ID 175212130 (and rated our vulnerability as the second highest severity among the total five-levels) and rewarded us with a bug bounty, Apple assigned an internal vulnerability-ID 755406462, Microsoft assigned an internal case tracking-ID MSRC-63104, and TI informed us that they are tracking the Bluetooth SIG's recommendations for fixing this vulnerability.

The BAT Attacks	Cost				Difficulty	Impacts
	Involved Malicious Devices	Uninterrupted Passively Monitoring	Keeping Actively Probing	Tunnelling	(Requiring Central and Peripheral to be Nearby)	
Passive Attacks (§4)	Sniffer	✓	✗	✗	✓	❶
Active Attacks from Malicious Centrals (§5.1)	Sniffer, Central, Peripheral	✗	✓	✓&✗	✗	❶,❷,❸
Active Attacks from Malicious Peripherals (§5.2)	Sniffer, Peripheral	✗	✗	✗	✗	❶,❷,❸

Table 2: Summary of Our Attacks. ❶ means "monitoring the user's behavior", ❷ means "tracking a victim's past trajectory", and ❸ means "tracking a victim's real time location".

9 RELATED WORK

Bluetooth Privacy. Over the past 20 years, numerous Bluetooth device tracking attacks have been proposed by collecting the advertising packets (e.g., ADV_IND, SCAN_REQ) using sniffers [1–5, 13, 14, 25–27, 35–37], as summarized in Table 3. Among them, earlier efforts such as BlueTrack [13] and BLEB [14] track the Bluetooth devices that use public addresses. Marco et al. [27] track the Bluetooth classic devices that do not use address randomization via the information leaked from the frame encoding.

However, when the devices use randomized MAC addresses, although intuitively one can associate a disappeared address and a newly appeared address to break the address randomization, one location can have multiple devices, which can appear or disappear simultaneously (e.g., turning on/off devices, devices moving out/in the range of sniffers). As such, monitoring the devices solely will have high false alerts, and extra efforts are needed for the tracking. To this end, Ludant et al. [26] linked and tracked the BLE addresses via the collection of Bluetooth Classic (BT) Addresses. Our attacks instead do not need to collect both Bluetooth classic packets (which are hard to be collected as discussed in [27]) and BLE packets, and we only need to collect BLE advertising packets.

There are also tracking attacks against specific implementations of Bluetooth devices (e.g., Apple devices [36–39] and wearable fitness trackers [40]). Some attacks exploit static payloads (e.g., fixed manufacture identifiers [2, 3], fixed information elements [41, 42], and fixed GATT attribute [43])) to track devices. Most of those attacks are implementation specific (which usually requires unchanged payload), whereas our BAT attacks are based on protocol level flaws we identified. There are also tracking attacks requiring the attacker to obtain keys (e.g., [25, 44, 45]). Our attacks do not make such assumption. Moreover, unlike the traditional tracking attacks [3, 18], where the attacker has to collect many BLE packets and analyze them one by one to identify the victim, our attack from malicious peripherals, for example, replays an old packet and the victim will be exposed instantly.

Bluetooth Security. Bluetooth are also subject to multiple security attacks such as brute force attacks against PIN (e.g., [46, 47]), eavesdropping attacks (e.g., [48–50]), MiTM attacks [51–54], and vulnerable pairing (e.g., [23, 55, 56]). In addition to these link layer attacks, there are also efforts targeted on Bluetooth applications. For example, the Bluetooth Misbonding Attacks [21] target peripherals that do not enforce application layer authentication and BadBluetooth [20] exploits the weakness when the central does not authenticate the peripherals. Compared to these works, we are the first to exploit the allowlist in Bluetooth protocol for device tracking.

Prior attacks in wireless sensor networks could also be applied to Bluetooth. For instance, in our Attack Example III, we have leveraged the concept of the wormhole attacks [29] to establish a tunnel between a sniffer and the malicious central located nearby of the stationary peripheral. While wormhole attack alone could be used to tunnel connections between a central and a peripheral to identify a device, it would require additional information such as our allowslisting side channels when there are multiple devices nearby.

Attacks	Non-Implementation Specific	Against RPA	w/o Unchanged Payloads	w/o Installing Malware	w/ either BLE or BT Packets	w/o Stealing IRK/Key
Jakobsson et al. [1]	✗	✗	✗	✓	✓	✓
BlueTrack [13]	✗	✗	✗	✓	✓	✓
BLEB [13]	✗	✗	✗	✓	✓	✓
BLE-Guardian [2]	✗	✓	✗	✓	✓	✓
Korolova et al. [5]	✗	✓	✗	✗	✓	✓
Becker et al. [3]	✗	✓	✗	✓	✓	✓
Martin et al. [37]	✗	✓	✗	✓	✓	✓
Celosia et al. [4]	✗	✓	✗	✓	✓	✓
BLEScope [19]	✗	✗	✗	✓	✓	✓
Stute et al.(2019) [39]	✗	✓	✗	✓	✓	✓
Marco et al. [27]	✗	✗	✓	✓	✓	✓
Zhang et al. [25]	✗	✓	✓	✓	✓	✗
FirmwXray [35]	✗	✗	✓	✓	✓	✓
Guillaume et al. [36]	✗	✓	✗	✓	✓	✓
Cominelli et al. [27]	✗	✗	✗	✓	✓	✓
Stute et al.(2021) [38]	✗	✓	✗	✓	✓	✓
Heinrich et al. [44]	✗	✓	✓	✓	✓	✗
Ludant et al. [26]	✗	✓	✗	✓	✗	✓
BAT Attacks	✓	✓	✓	✓	✓	✓

Table 3: Comparison of our BAT attacks with others.

10 CONCLUSION

We have presented BAT attacks by exploiting the single-bit of side channel of response and no-response from BLE devices with allowlist to track their randomized MAC addresses. Our attacks can be launched passively, or actively due to the vulnerability of current RPA generation and resolution algorithm. We have also presented defense SABLE, by adding a random sequence number when generating and resolving the RPA-type MAC addresses to defeat our active BAT attacks, and enforcing an interval unpredictable, central and peripheral synchronized RPA generation scheme to counter passive attacks. The experimental results show that our defense has negligible impact on both the battery consumption and performance overhead for the involved centrals and peripherals.

ACKNOWLEDGEMENTS

We would like to thank Shuo Chen, Syed Rafiul Hussain, Moti Yung, and the anonymous reviewers for their helpful comments on an early draft of this paper. This research was partially supported by NSF award 2112471.

REFERENCES

[1] M. Jakobsson and S. Wetzel, "Security weaknesses in bluetooth," in *Cryptographers' Track at the RSA Conference.* Springer, 2001, pp. 176–191.
[2] K. Fawaz, K.-H. Kim, and K. G. Shin, "Protecting privacy of ble device users," in *25th USENIX Security Symposium (USENIX Security 16)*, 2016, pp. 1205–1221.

[3] J. K. Becker, D. Li, and D. Starobinski, "Tracking anonymized bluetooth devices," *Proceedings on Privacy Enhancing Technologies (PETS)*, vol. 2019, no. 3, pp. 50–65, 2019.

[4] G. Celosia and M. Cunche, "Saving private addresses: an analysis of privacy issues in the bluetooth-low-energy advertising mechanism," in *Proceedings of the 16th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, 2019, pp. 444–453.

[5] A. Korolova and V. Sharma, "Cross-app tracking via nearby bluetooth low energy devices," in *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy (CODASPY)*, 2018, pp. 43–52.

[6] S. Bluetooth, "Bluetooth core specification version 4.2," *Specification of the Bluetooth System*, 2014.

[7] Apple Inc., "Accessory Design Guidelines for Apple Devices) ," https://developer.apple.com/accessories/Accessory-Design-Guidelines.pdf, 2019.

[8] S. Bluetooth, "Bluetooth core specification version 4.1," *Specification of the Bluetooth System*, 2011.

[9] ——, "Bluetooth core specification version 5.2," *Specification of the Bluetooth System*, 2020.

[10] Google, "Android open source project," https://source.android.com/.

[11] Nordic, "Bluetooth programming framework," https://embeddedcentric.com/nordic-ble-training/.

[12] Bluetooth-SIG, "Bluetooth core specification version 4.0," *Specification of the Bluetooth System*, 2010.

[13] M. Haase, M. Handy *et al.*, "Bluetrack–imperceptible tracking of bluetooth devices," in *Ubicomp Poster Proceedings*, vol. 2, 2004.

[14] T. Issoufaly and P. U. Tournoux, "Bleb: Bluetooth low energy botnet for large scale individual tracking," in *2017 1st International Conference on Next Generation Computing Applications (NextComp)*. IEEE, 2017, pp. 115–120.

[15] Amazon, "Pro wireless controller for nintendo switch bluetooth controllers gamepad pc joystick controller for switch controllers with dual vibrations and motion sensors," https://www.amazon.com/Bluetooth-Wireless-NS-Switch-Joystick-Controller/dp/B08FMH783R/.

[16] Google, "Android 6.0 changes," https://developer.android.com/about/versions/marshmallow/android-6.0-changes#behavior-hardware-id, 2016.

[17] C. Ellis, H. Wen, Z. Lin, and A. Arora, "Replay (far) away: Exploiting and fixing google/apple exposure notification contact tracing," *Proceedings on Privacy Enhancing Technologies (PETS)*, vol. 2022, no. 4, 2022.

[18] C. Matte, M. Cunche, F. Rousseau, and M. Vanhoef, "Defeating mac address randomization through timing attacks," in *Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, 2016, pp. 15–20.

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

[20] F. Xu, W. Diao, Z. Li, J. Chen, and K. Zhang, "Badbluetooth: Breaking android security mechanisms via malicious bluetooth peripherals," in *Proceedings of the 26th Annual Network and Distributed System Security Symposium (NDSS'19), San Diego, CA*, 2019.

[21] M. Naveed, X. Zhou, S. Demetriou, X. Wang, and C. A. Gunter, "Inside job: Understanding and mitigating the threat of external device mis-binding on android," in *21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February 23-26, 2014*, 2014.

[22] L. H. NEWMAN, "Sneaky zero-click attacks are a hidden menace," https://www.wired.com/story/sneaky-zero-click-attacks-hidden-menace/, April 2020, (Accessed on 12/24/2021).

[23] D. Antonioli, N. O. Tippenhauer, and K. Rasmussen, "Bias: Bluetooth impersonation attacks," in *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2020.

[24] D. Antonioli, N. O. Tippenhauer, and K. B. Rasmussen, "The {KNOB} is broken: Exploiting low entropy in the encryption key negotiation of bluetooth br/edr," in *28th {USENIX} Security Symposium ({USENIX} Security 19)*, 2019, pp. 1047–1061.

[25] Y. Zhang, J. Weng, R. Dey, Y. Jin, Z. Lin, and X. Fu, "Breaking secure pairing of bluetooth low energy using downgrade attacks," in *29th {USENIX} Security Symposium ({USENIX} Security 20)*, 2020, pp. 37–54.

[26] N. Ludant, T. D. Vo-Huu, S. Narain, and G. Noubir, "Linking bluetooth le & classic and implications for privacy-preserving bluetooth-based protocols," in *2021 IEEE Symposium on Security and Privacy (SP)*, 2021.

[27] M. Cominelli, F. Gringoli, P. Patras, M. Lind, and G. Noubir, "Even black cats cannot stay hidden in the dark: Full-band de-anonymization of bluetooth classic devices," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 534–548.

[28] SENA, "Patch antenna - rp-sma-r/a right-hand thread, 9dbi," http://www.senanetworks.com/pat-g01r.html.

[29] Y.-C. Hu, A. Perrig, and D. B. Johnson, "Wormhole attacks in wireless networks," *IEEE journal on selected areas in communications*, vol. 24, no. 2, pp. 370–380, 2006.

[30] Statistics, "Bluetooth device shipments worldwide from 2015 to 2026," https://www.statista.com/statistics/1220933/global-bluetooth-device-shipment-forecast/, April 2020, (Accessed on 12/24/2021).

[31] Adafruit , "Adafruit sniffer," https://learn.adafruit.com/introducing-the-adafruit-bluefruit-le-sniffer/introduction.

[32] Ganesh, "The current psoc ez-serial fw does not support rpa for whitelist," https://community.cypress.com/thread/51728?start=0&tstart=0, 2020.

[33] S. Labs, "Twhitelisting (silcon labs official document)," https://docs.silabs.com/bluetooth/3.0/general/adv-and-scanning/whitelisting, 2020.

[34] E. Garcia-Espinosa, O. Longoria-Gandara, I. Pegueros-Lepe, and A. Veloz-Guerrero, "Power consumption analysis of bluetooth low energy commercial products and their implications for iot applications," *Electronics*, vol. 7, no. 12, p. 386, 2018.

[35] H. Wen, Z. Lin, and Y. Zhang, "Firmxray: Detecting bluetooth link layer vulnerabilities from bare-metal firmware," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2020.

[36] G. Celosia and M. Cunche, "Discontinued privacy: Personal data leaks in apple bluetooth-low-energy continuity protocols," *Proceedings on Privacy Enhancing Technologies*, vol. 2020, no. 1, pp. 26–46, 2020.

[37] J. Martin, D. Alpuche, K. Bodeman, L. Brown, E. Fenske, L. Foppe, T. Mayberry, E. Rye, B. Sipes, and S. Teplov, "Handoff all your privacy–a review of apple's bluetooth low energy continuity protocol," *Proceedings on Privacy Enhancing Technologies*, vol. 2019, no. 4, pp. 34–53, 2019.

[38] M. Stute, A. Heinrich, J. Lorenz, and M. Hollick, "Disrupting continuity of apple's wireless ecosystem security: New tracking,{DoS}, and {MitM} attacks on {iOS} and {macOS} through bluetooth low energy,{AWDL}, and {Wi-Fi}," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 3917–3934.

[39] M. Stute, S. Narain, A. Mariotto, A. Heinrich, D. Kreitschmann, G. Noubir, and M. Hollick, "A billion open interfaces for eve and mallory:{MitM},{DoS}, and tracking attacks on {iOS} and {macOS} through apple wireless direct link," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 37–54.

[40] A. K. Das, P. H. Pathak, C.-N. Chuah, and P. Mohapatra, "Uncovering privacy leakage in ble network traffic of wearable fitness trackers," in *Proceedings of the 17th International Workshop on Mobile Computing Systems and Applications*. ACM, 2016, pp. 99–104.

[41] M. Vanhoef, C. Matte, M. Cunche, L. S. Cardoso, and F. Piessens, "Why mac address randomization is not enough: An analysis of wi-fi network discovery mechanisms," in *Proceedings of the 11th ACM on Asia conference on computer and communications security*, 2016, pp. 413–424.

[42] H. Wen, Q. Zhao, Z. Lin, D. Xuan, and N. Shroff, "A study of the privacy of covid-19 contact tracing apps," in *International Conference on Security and Privacy in Communication Networks*, 2020.

[43] G. Celosia and M. Cunche, "Fingerprinting bluetooth-low-energy devices based on the generic attribute profile," in *Proceedings of the 2nd International ACM Workshop on Security and Privacy for the Internet-of-Things*, 2019, pp. 24–31.

[44] A. Heinrich, M. Stute, T. Kornhuber, and M. Hollick, "Who can devices? security and privacy of apple's crowd-sourced bluetooth location tracking system," *Proceedings on Privacy Enhancing Technologies*, vol. 2021, no. 3, pp. 227–245, 2021.

[45] A. Heinrich, M. Stute, and M. Hollick, "Openhaystack: a framework for tracking personal bluetooth devices via apple's massive find my network," in *Proceedings of the 14th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, 2021, pp. 374–376.

[46] A. Becker and I. C. Paar, "Bluetooth security & hacks," *Ruhr-Universität Bochum*, 2007.

[47] Y. Shaked and A. Wool, "Cracking the bluetooth pin," in *Proceedings of the 3rd international conference on Mobile systems, applications, and services*. ACM, 2005, pp. 39–50.

[48] D. Spill and A. Bittau, "Bluesniff: Eve meets alice and bluetooth." *WOOT*, vol. 7, pp. 1–10, 2007.

[49] M. Ryan, "Bluetooth: With low energy comes low security," in *Proceedings of the 7th USENIX Conference on Offensive Technologies*, ser. WOOT'13. Berkeley, CA, USA: USENIX Association, 2013, pp. 4–4. [Online]. Available: http://dl.acm.org/citation.cfm?id=2534748.2534754

[50] T. Rosa, "Bypassing passkey authentication in bluetooth low energy." *IACR Cryptology ePrint Archive*, vol. 2013, p. 309, 2013.

[51] D. Kügler, ""man in the middle" attacks on bluetooth," in *International Conference on Financial Cryptography*. Springer, 2003, pp. 149–161.

[52] K. Haataja and P. Toivanen, "Two practical man-in-the-middle attacks on bluetooth secure simple pairing and countermeasures," *IEEE Transactions on Wireless Communications*, vol. 9, no. 1, 2010.

[53] K. Hypponen and K. M. Haataja, ""nino" man-in-the-middle attack on bluetooth secure simple pairing," in *2007 3rd IEEE/IFIP International Conference in Central Asia on Internet*. IEEE, 2007, pp. 1–5.

[54] Y. Zhang, J. Weng, Z. Ling, B. Pearson, and X. Fu, "Bless: A ble application security scanning framework," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 636–645.

[55] D. Antonioli, N. O. Tippenhauer, and K. Rasmussen, "Low entropy key negotiation attacks on bluetooth and bluetooth low energy." *IACR Cryptol. ePrint Arch.*, vol. 2019, p. 933, 2019.

[56] ——, "Key negotiation downgrade attacks on bluetooth and bluetooth low energy," *ACM Transactions on Privacy and Security (TOPS)*, vol. 23, no. 3, pp. 1–28, 2020.