Johannes K Becker*, David Li, and David Starobinski

Tracking Anonymized Bluetooth Devices

Abstract: Bluetooth Low Energy (BLE) devices use public (non-encrypted) advertising channels to announce their presence to other devices. To prevent tracking on these public channels, devices may use a periodically changing, randomized address instead of their permanent Media Access Control (MAC) address. In this work we show that many state-of-the-art devices which are implementing such anonymization measures are vulnerable to passive tracking that extends well beyond their address randomization cycles. We show that it is possible to extract identifying tokens from the payload of advertising messages for tracking purposes. We present an address-carryover algorithm which exploits the asynchronous nature of payload and address changes to achieve tracking beyond the address randomization of a device. We furthermore identify an identity-exposing attack via a device accessory that allows permanent, non-continuous tracking, as well as an iOS side-channel which allows insights into user activity. Finally, we provide countermeasures against the presented algorithm and other privacy flaws in BLE advertising.

Keywords: Bluetooth, tracking, privacy, information leakage, side-channels, correlation attacks, traffic analysis.

DOI 10.2478/popets-2019-0036
Received 2018-11-30; revised 2019-03-15; accepted 2019-03-16.

1 Introduction

Bluetooth technology has facilitated the ubiquity of instant wireless connectivity, ranging from personal connected accessories, to smart homes, and localized and personalized, location-based shopping experiences. Since it was first adopted in mobile phones in the year 2000 [11], it has undergone five major Core Specification revisions with numerous amendments [10].

In early versions of the Bluetooth specification, the *permanent* Bluetooth MAC addresses of devices were regularly broadcasted in the clear, leading to major privacy concerns over the possibility of unwanted tracking [33]. This was addressed in the Bluetooth Core Specification 4.0 with the introduction of the *Bluetooth Low Energy (BLE)* standard also known as *Bluetooth Smart*. BLE allows device manufacturers to use temporary random addresses in over-the-air communication instead of their permanent address to prevent tracking [38]. However, these anonymization features are defined in a way that leaves a certain degree of flexibility to manufacturers. The optionality of such privacy protecting features is of special relevance, as the BLE standard was designed specifically to support low-energy devices such as smart watches and other wearable devices, which are an attractive target for adversarial tracking of their users.

BLE devices broadcast so-called *advertisements* on unencrypted, public channels in order to signal their presence to other devices. Ideally, this public broadcast contains all the required information to perform a device function, while not leaking unnecessary private information about the device or its user. In some cases, however, devices may broadcast data that exposes sensitive details about themselves or even other devices.

In this work, we show how even state-of-the-art devices such as Windows 10 computers and iPhones, which *do* implement privacy protecting measures such as address randomization may be vulnerable to continuous tracking. We first examine various types of advertising messages and identify so-called *identifying tokens*, which are unique to a device and remain static for long enough to be used as secondary identifiers besides the address. We present an online algorithm called the **address-carryover algorithm**, which exploits the fact that identifying tokens and the random address do not change in sync, to continuously track a device despite implementing anonymization measures. To our knowledge, this approach affects *all Windows 10, iOS, and macOS devices*. The algorithm does not require message decryption or breaking Bluetooth security in any way, as it is based entirely on public, unencrypted advertising traffic.

The Bluetooth 5 Specification extends usable communication range to whole buildings or hundreds of meters in line-of-sight transmissions [9, 26]. While the tracking attack proposed in this paper considers tracking by a single adversary in such an operating radius,

***Corresponding Author: Johannes K Becker:** Boston University, E-mail: jkbecker@bu.edu
David Li: Boston University, E-mail: dlhi@bu.edu
David Starobinski: Boston University, E-mail: staro@bu.edu

previous work [22] suggests that local BLE tracking methods may be significantly compounded by coordinating them in a botnet of adversaries, resulting in potentially global tracking capabilities.

The main contributions of this paper are as follows:

1. We describe a tracking vulnerability that affects *Windows 10, iOS, and macOS devices* as long as they are continuously observed by the adversary.
2. We develop a methodology that can be applied to devices from various manufacturers, based on raw BLE advertising log files.
3. We present an algorithm that allows tracking beyond the address randomization of a device, and measure the resulting maximum tracking time (MTT).
4. We identify other privacy vulnerabilities that exist on certain device types, which expose device identifiers permanently via a peripheral, and which leak activity information on iOS devices.
5. We provide recommendations and potential countermeasures to the tracking vulnerabilities uncovered in this work.

The rest of this paper is structured as follows: Section 2 discusses prior related work. Section 3 presents background information on the Bluetooth protocol necessary to understand this work. In Section 4, we describe our adversarial model and the methodology used in this work, followed by the experimental setup in Section 5. We present our results in Section 6, followed by recommendations for the avoidance of unwanted device tracking in Section 7. We summarize our findings and give an outlook on further research in Section 8.

2 Related Work

Privacy and security concerns over Bluetooth date back to its very first release [23]. Anonymizing devices in public channel communication only became available with the introduction of BLE in Bluetooth 4.0 [6]. A lot of research regarding the effectiveness of MAC address randomization is focused on Wi-Fi, where the same privacy concern of broadcasting permanent identifiers exists, but vulnerabilities are often not easily transferable to the Bluetooth case as they are based on different areas specific to the Wi-Fi network stack. We will highlight some important works relating to more general cases of Bluetooth and Wi-Fi tracking concerns, as well as more BLE-specific techniques and utilities.

In 2007, Spill and Bittau [33] presented several techniques for eavesdropping on Bluetooth 2.0 communication using a GNU Radio-based Bluetooth sniffer and USRP software-defined radio hardware. Their work describes an approach for intercepting packets, and reverse-engineering all the parameters required to eavesdrop on Bluetooth communication [33]. However, these findings only concern the Bluetooth Classic implementation, which is of decreasing relevance in light of BLE and Bluetooth 5.

In 2015, Jameel and Dungen presented an open-source library for scanning Bluetooth Low Energy (LE) and Active RFID advertising [24]. Their work summarizes different available Beacon protocols, which are proximity-based broadcast protocols [1, 18] and enable all kinds of localized interactions with smartphones and other Bluetooth devices via the BLE advertising channels. Furthermore, the authors published a library called *advlib* [29], which processes raw BLE advertising messages and decodes them into an open, portable data format. This library enables software developers to easily integrate BLE advertising-based functionality into their software, without having to manually decode low-level protocols. The library further powers the open-source "collaborative repository" *Sniffypedia* [30], which presents a large number of publicly known BLE advertising identifiers in a searchable and accessible format. This platform can help classify Bluetooth device classes for reconnaissance purposes, but does not offer device tracking capabilities.

Vanhoef et al. [34] present techniques to gain access to permanent MAC addresses by exploiting probe requests in Wi-Fi. They develop an algorithm which relies on timing features and sequence numbers found in Wi-Fi probe requests to identify devices regardless of their MAC address. They further describe a variant of the *Karma Attack* – exploiting the fact that many devices will expose information to supposedly known and trusted networks [13] by creating a catch-all access point [14] – which creates large numbers of popular Wi-Fi networks in order to invite devices to connect, often presenting their permanent MAC address in a supposedly trusted context.

Issoufaly and Tournoux [22] show that despite the existence of privacy-preserving MAC address randomization in Bluetooth 4.0 LE, not all devices make use of this functionality and are therefore vulnerable to tracking. Furthermore, they showed how maliciously distributing suitable tracking software to a number of

mobile devices – a "BLE Botnet" – extends tracking capabilities far beyond the local transmission range of regular Bluetooth communication. Combining BLE privacy with the notion of a potentially global botnet motivates our work as it extends the privacy threat from local presence detection to large-scale location tracking, which would be realistic to implement for nation-state actors. While Issoufaly and Tournoux focus on tracking devices that use static device addresses, our work focuses on tracking devices that use dynamically randomized addresses. These addresses change depending on regeneration parameters set by their manufacturers.

Along the same line, Martin et al. [27] show that MAC address randomization is often defeated by not being implemented properly or consistently, partly building upon previous approaches [34]. Furthermore, they show flaws in the address randomization of Android phones, which allow the deduction of device types via address prefixes, as well as their global MAC address via information found in broadcasted WPS attributes. They also identify a control frame attack that exposes the permanent Wi-Fi MAC address of a device with 100% success rate [27, p. 280]. Our approach differs in that we do not retrieve information through reversal of hashes or other reverse-engineering methods, and we use entirely passive sniffing to generate insights which allow device tracking. Our tracking method, based on extracting identifying tokens in the payload of advertising messages, leverages specific Bluetooth features and differs from this previous line of work.

3 Background

Bluetooth is a wireless communication protocol on the 2.4 GHz ISM band. Our focus lies on the *Low Energy* specification, which was introduced in 2010 with Bluetooth 4.0 [6]. Bluetooth LE (BLE) is optimized for small, often battery-powered devices with relatively strict energy restrictions, like connected wearables, wireless smart sensors, or computer accessories like digital pencils. The Bluetooth 5 Specification adds various feature and performance improvements, notably a larger communication range [9], which allows for up to 780 meters of line-of-sight connectivity [26].

We focus on introducing the elements that are essential for the experiments described in the following Sections. We refer to Heydon [20, p. 7] for a general high-level introduction to the Bluetooth 5 architecture and the Bluetooth 5 Core Specification for details [8].

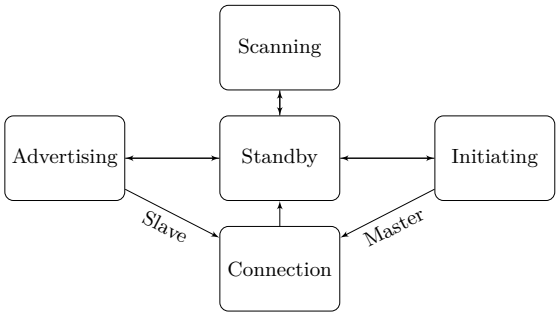

Fig. 1. The Link Layer State machine [20, p. 15].

3.1 Physical Layer and Link Layer

BLE operates on 40 physical channels spaced in 2 MHz steps from 2402 MHz to 2480 MHz [8, p. 2535f.]. Three of these channels are so-called *advertising channels* at 2402 MHz, 2426 MHz, and 2480 MHz – frequencies with minimal interference with other 2.4 GHz band traffic (notably Wi-Fi) [24]. These channels are used to broadcast *advertising messages*, which may be simple periodic, presence announcements, scanning requests towards other devices, beacons containing location-based metadata, and other public broadcast-type information. The remaining channels are used primarily for data transmissions between paired devices and employ a pseudo-random channel hopping scheme, which is negotiated between the two connected parties upon connection to avoid interference with other ISM band traffic.

On the link layer, Bluetooth defines different states, shown in Figure 1. In the *Advertising* state, the link layer creates *Advertising Events* which are broadcasted on all advertising channels. These events contain different types of Payload Data Units (PDU)[1]. In our work, the following PDU types are relevant:

- the *Connectable Directed Event* using the `ADV_DIRECT_IND` PDU, which invites a specifically addressed device to initiate a connection.
- the *Connectable and Scannable Undirected Event* using the `ADV_IND` PDU, which allows any device to respond with scanning requests or connection requests.

In the *Scanning* state, the link layer receives advertising messages without responding (*passive scanning*) or responds with *scan requests* to inquire about further

1 The full list of advertising event types and corresponding PDU types can be found in [8, p. 2609, Table 4.1].

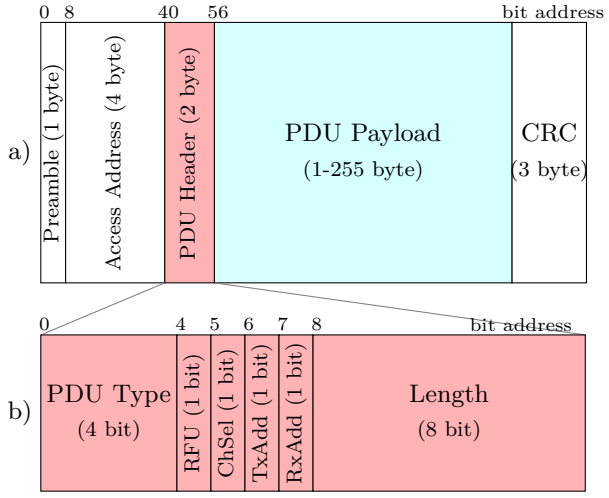

Fig. 2. The BLE advertising packet format (Fig. 2.a)) and the Advertising Channel PDU Header (Fig. 2.b)) [8, p. 2562, 2567].

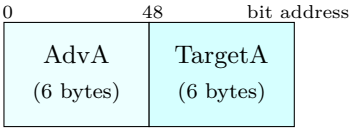

Fig. 3. The PDU Payload format for the `ADV_DIRECT_IND` PDU Type contains an Advertising Address (AdvA) and a Target Address (TargetA). [8, p. 2570.].

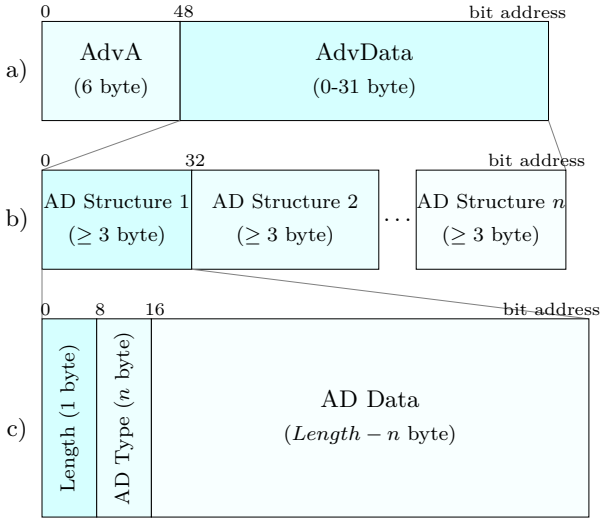

Fig. 4. PDU Payload format for `ADV_IND` and `ADV_NONCONN_IND` PDU Types (Fig. 4.a)), the *AdvData* format (Fig. 4.b)), and the *AD Structure* format (Fig. 4.c)) [8, p. 2086, 2569ff.] [4].

device details (*active scanning*) [8, p. 2633]. In the *Initiating* state, the link layer listens for advertising messages, and sends *Connect Requests* to advertising devices. If the requested device responds, the devices can enter a *Connection* state [8, p. 2637].

In the Connection State, devices implementing the so-called *Central* role can connect to up to 8 *Peripheral* role devices as their Master, while Peripheral devices can connect to one Central role device as a Slave. Smartphones or PCs are typical Central role devices connecting to multiple wireless input devices, activity trackers or other wireless sensors. However, devices may implement both roles and may therefore send advertising messages (Peripheral role) while scanning for advertising messages (Central role)[2]. In our work, we observe devices which are in the Advertising state using an observer device which is in the Passive Scanning state.

3.2 Advertising Packet Format

We next provide details on the structure and contents of advertising packets, since those will be exploited to track and distinguish between different devices. Advertising packets follow a structure shown in Figure 2.a), consisting of a preamble, an Access Address (AA)[3], a Payload Data Unit (PDU) and a CRC field. The advertising channel PDU itself consists of a header and a payload depending on the PDU Type defined in the header [8, p. 2567] (see Figure 2.b)).

The PDU Payload of a *directed* advertising event contains an *Advertising Address (AdvA)* and a *Target Address (TargetA)*, as shown in Figure 3. Only devices with the matching Target Address can initiate a connection by sending a Connect Request.

Indirected advertising allows any device receiving the PDU to respond with a Scan Request (requesting information about available features) or a Connect Request. The PDU Payload for this type of advertising is formatted as shown in Figure 4, containing an *Advertising Address (AdvA)* and *Advertising Data (AdvData)*, which may contain one or more *AD Structures* (see Figures 4.b). Finally, *AD Types* define the content of the *AD Data* contained in each AD Structure (Figure 4.c).

An overview of some AD Types that we will use later in this paper is shown in Table 1, we refer to [4] for the full list.

Most AD Types define simple numbers or strings. The *Manufacturer Specific Data* type is worth highlighting, as it gives device manufacturers the flexibility to

2 I.e., a BLE device can be in more than one state shown in Figure 1 at the same time.

3 The access address is always `0x8E89BED6` for advertising payloads [8, p. 2563].

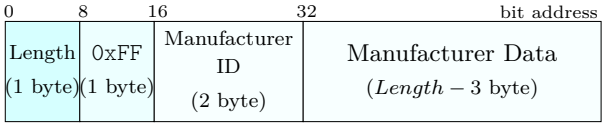

Fig. 5. The *Manufacturer Specific Data* AD Structure format [3, 7].

AD Type	Name	Description
0x01	Flags	BLE capability flags
0x06	List of 128-bit Service Class UUIDs	globally unique identifier for a certain device service
0x09	Complete Local Name	Device "friendly name"
0x16	Service Data	Service UUID as per [5]
0xFF	Manufacturer Specific Data	custom data structure container (see Figure 5)

Table 1. An overview of some relevant AD Types [4] [8, Vol. 3, Part C, Sections 8.1, 11.1, and 18] [7, Part A, Section 1].

define further data structures within the AD Data field. Figure 5 shows a Manufacturer Specific Data AD Structure. This AD Type allows manufacturers to develop their own advertising-based protocols that live within this data[4], which may contain data that our proposed algorithm exploits.

3.3 Advertising Addresses

The Bluetooth architecture generally refers to devices that transmit advertising packets as *advertisers*. Devices listening to advertising packets are referred to as *scanners* if they listen without intention of connecting, and as *initiators* if they intend to make Connect Requests in response to suitable advertising events [8, p. 170].

The Advertising Address shown in Figures 3 and 4 is the 48-bit MAC address of the advertiser transmitting the advertising packet. This address may be a *public* or *random* address depending on the *TxAdd* flag in the PDU Header (see Fig. 2).

A public address (TxAdd = 0) is the standard IEEE-assigned 48-bit MAC address obtained from the IEEE Registration Authority, in which the 24 least significant bits are manufacturer-assigned, while the 24 most significant bits represent the IEEE-assigned company ID [21]. These addresses are permanently attached to the device hardware, making them a vulnerability to long term tracking if used in public communication.

Random addresses (TxAdd = 1) may be *static*, in which case they are randomly generated whenever the device reboots or remain with the device permanently, which makes them vulnerable to long-term tracking. *Random private addresses* prevent address-based tracking by regenerating periodically, essentially anonymizing the device identity [38]. Such random private addresses may either be an *unresolvable* address made of a random number, or they may be *resolvable* to a permanent address by trusted devices holding an *Identity Resolving Key (IRK)*. This can be used to send directed advertising to a specific device without broadcasting its permanent address in the clear and is further discussed in Section 7 containing our Recommendations. Resolvable and unresolvable random addresses can be distinguished by their two most significant bits being set to 0b01 and 0b00, respectively [8, p. 2558].

4 Methodology

4.1 Adversarial Model

This paper aims to uncover potential security and privacy flaws in BLE devices, as they appear "in the wild". We consider a *local adversary*[5] in the *scanning* link layer state (see Section 3.1). As such, it is a *passive*, *external* adversary which reads continuously, but never adds, removes, or modifies ongoing BLE adversary traffic. Specifically, the adversary never sends scan requests or connect requests, and remains entirely passive during the attack, i.e., there is no bi-directional interaction with the target device. We assume that the adversary is capable of monitoring the ongoing advertising traffic in order to be aware of changes in advertising messages as they occur over time.

We do not make any assumptions about the victim devices, other than Bluetooth being turned on. Note that devices may implement the Peripheral and Central roles at the same time, and thus remain in advertising state for the Peripheral role despite ongoing connections as a Master device. Thus, the ability of tracking smartphones or computers via passive tracking mainly depends on whether they implement the Peripheral role.

4 Widely adopted manufacturer-specific data protocols are the Apple iBeacon protocol [1] or other open protocols like Google's Eddystone [18] and Radius Networks' AltBeacon [28].

5 Emphasized terms in this Section in line with the definitions of privacy metrics given in [37, Section 4].

The adversary only needs to observe Bluetooth advertising messages (which are sent in plain text), i.e., the adversary does not require the capability to read encrypted data transmissions or follow channel hopping. The adversary implements a sniffing device (in our case, a Software-Defined Radio (SDR)) which listens to one of the Bluetooth advertising channels (i.e., channels 37, 38, or 39[6]). The packets are then broken down into their elements as defined in the Bluetooth specification (see also Figure 2) and further analyzed for any *observable data* which could be used to compromise user privacy.

As advertising messages are sent in the clear and are not authenticated, proving authenticity and integrity of an advertising message is not easy in most cases. As such, the methodology used in this paper considers benign communication by devices which are oblivious to potential privacy issues in their advertising patterns, i.e., they neither forge the contents of their messages nor change the traffic patterns for their messages in order to obfuscate their identity.

This work assumes that an adversary will at some point make a connection between an arbitrary identifier in an advertising event and its real-world associated device, and will then continuously observe ongoing BLE advertising traffic in order to track it without the user's consent for the longest time period possible. *Adversary success* is defined as the adversary being able to hold on to a device identity beyond its address randomization using local, passive, continuous observation.

In this paper, we use the terms *user* and *device* synonymously. We assume that devices are being tracked with the intent of tracking their users, and the ability to track a device implies the ability to track its user. We therefore do not consider devices that are shared between multiple users and would require such a logical distinction.

4.2 Privacy Metrics

The applied metrics focus on the adversary's ability to track a device over time due to some observable signature. Specifically, let the *anonymity set size* AS_u represent the set of users that an adversary cannot distinguish from a targeted user u [37, p. 57:9] [32, p. 5], i.e.:

$$|AS_u| = \text{number of users indistinguishable from user } u. \quad (1)$$

The primary metric of interest in this paper is the Maximum Tracking Time (MTT), which is defined as the cumulative time that a user u can be uniquely identified [37, p. 57:30] [32, p. 5], that is:

$$MTT_u = \text{time during which } |AS_u| = 1 \text{ for a user } u. \quad (2)$$

We show that several types of devices have unbounded MTT.

4.3 Device Tracking Algorithm

We propose a two-phase algorithm which allows tracking of devices beyond their address randomization. The algorithm consists of an offline *pre-processing* phase and an online *tracking phase*.

4.3.1 Phase 1: Pre-Processing

The pre-processing phase uses recorded log files containing decoded BLE advertising packets (see Figure 2) and their PDU Payloads in raw format. All collected log files are processed offline in order to identify potential *identifying tokens* occurring in advertising messages. An identifying token can be any sequence of bits contained in an advertising message which allows distinguishing one device from another – be it by design (such as the advertising address) or by a side effect. As a general rule, an identifying token for a given device should remain fixed over a certain period of time, while being unique to one device (i.e., its values should not collide with values produced by other devices within range over time).

The set of useful identifying tokens for a class of devices depends on its operating system and other device-specific hardware or software features, as all of these factors determine the structure and content of advertising messages.

In this sense, a universal MAC address [21, p. 22f.] is a perfect identifying token as it is guaranteed to be unique to a device, and is permanently linked to a device. In a local context, these requirements can be relaxed as long as the distinction between the devices in range is still possible with high confidence.

6 It is assumed that any of the available advertising channels can be considered equivalent as advertising events are usually broadcast to all three channels in a round-robin fashion.

Identifying tokens may be found either by looking at the raw PDU Payload and isolating a byte sequence which fulfills the requirements laid out above, or by breaking down the PDU Payload according to the Bluetooth protocol specification and identifying suitable metadata elements.

In order to avoid collisions between identifying tokens, it is important that the bit length n of the identifying token be large enough. Formally, suppose an adversary aims to track a Bluetooth device over a certain period, say a week, in an area containing at most m other Bluetooth devices with the same types of identifying tokens (devices with different identifying tokens are trivial to distinguish). Furthermore, suppose that each device changes its identifying token at a rate no faster than once every 15 minutes (based on our experimental data, 15 minutes appears to be an upper bound on the frequency of changes of identifying tokens). This means that the targeted device will generate at most $k = 672$ tokens in a week.

Assume that each identifying token is generated uniformly at random (a reasonable assumption based on our experimental data). The probability that another Bluetooth device produces tokens that do not conflict with one of these k tokens is at least $(1 - k/2^n)^k$. Hence, the probability that all the other m devices produce different tokens than the targeted device is at least $(1 - k/2^n)^{mk} \geq 1 - mk^2/2^n$, where the latter inequality is obtained by applying Bernoulli's inequality [17]. Therefore, by choosing $1 - mk^2/2^n \geq 1 - p$ or $n \geq \lceil \log_2(mk^2/p) \rceil$, we can ensure a collision probability smaller than p. As an example, for $m = 1000$ devices and $p = 10^{-3}$, we obtain $n \geq 39$ bits. Consequently, identifying tokens that have a length of at least 40 bits (i.e., 5 bytes), satisfy this inequality.

Note that performing pre-processing is only required once per device class. Once a suitable set of identifying tokens has been found for a class of devices, it can be applied to track any individual device in this class.

4.3.2 Phase 2: Tracking

The **address-carryover algorithm** (Algorithm 1) is an online algorithm that continuously observes changes in the address as well as any other relevant identifying tokens found in the pre-processing phase.

The algorithm listens to incoming advertising messages m_i as they are broadcast on one of the BLE advertising channels with the goal of tracking a target device D beyond the point where it randomizes its advertising address.

If the incoming message contains the right identifying tokens for the class of devices that D belongs to, the tokens are extracted (line 4). Then, if the incoming advertising address is identical to the existing advertising address (*addr*), we still know the device identity and we update the stored values of the identifying tokens if any changes are detected (line 7). Otherwise, if the advertising address has changed, a match is attempted using any of the available identifying tokens as a pseudo-identity – in case of a successful match, the address of device D can be updated with the incoming address and address-carryover was successful (line 11). In either of these cases, the *timeout* counter is reset to indicate that the device D is being tracked successfully. If neither address nor identifying token matches succeed for a certain period of time T_{max}, it is assumed that device tracking has failed and the algorithm terminates.

The algorithm requires an amount of memory and computation that are linear in the size of the set of identifying tokens. Since there is only a handful of identifying tokens, the algorithm can be run in real-time without limitation.

5 Experimental Setup

The experimental setup used in this work consists of monitoring one of the BLE advertising channels for a certain amount of time and analyzing the content of advertising events that are broadcast by devices within typical Bluetooth proximity.

We use a modified version of Xianjun Jiao's SDR-based BLE sniffer [25] for all experiments. The collected log files contain advertising messages according to Figure 2, with all metadata decoded and the PDU payload in raw hexadecimal ASCII characters. In the pre-processing phase, we decode them according to the structure shown in Figures 2, 4, and 5, where applicable. The system used for all experiments is based on a Ubuntu Xenial-based PC and the HackRF One SDR.

We chose to use an SDR-based Bluetooth sniffer over a sniffer based on the regular Bluetooth stack to remain independent of variations in Bluetooth implementations on different hardware and OSes, to be able to observe raw data which is typically discarded by the low-level network stack, and in order to facilitate future consideration of physical layer metrics in our research.

Algorithm 1: Address-carryover algorithm

Input : stream of advertising messages $m_1, m_2, \ldots$
Variable : *addr* is an advertising address for device D
Variable : A is a list of identifying tokens $[a_1, a_2, ...]$ for device D
Variable : T_{max} is the maximum time tracking will be attempted
Variable : *timeout* is a time counter
Initialize: *addr* to the current random address of the target device D, A to the currently broadcast values of A for device D. Set *timeout* to zero and begin counting.

```
1  while timeout < T_max do
2      receive message m_i;
3      if message m_i contains at least one identifying token then
4          extract incoming addr all contained identifying tokens from message m_i;
5          if incoming addr = existing addr then
6              if any incoming token values differ from existing token values in A then
7                  update A to reflect new values of tokens;
8              reset timeout ;                                  // D is known (address unchanged)
9          else unknown address:
10             if the value of any identifying token in m_i matches the corresponding value in A then
11                 addr := incoming address;
12                 update A to reflect new values of tokens;
13                 reset timeout;                               // D is known (address carried over)
                   Result: Successfully tracked device D, new address is <addr>.
14 end
   /* timeout exceeds T_max: D is lost                                                          */
```

However, advanced knowledge of SDR is not required for this paper.

Once we ensure that Bluetooth is enabled, the device is allowed to go to lock screen or be casually used, i.e., we do not require the Bluetooth settings to remain open during the attack. The sniffer passively listens to Bluetooth advertisements and does not actively engage in communication. We perform four weeks worth of 1-2 hour long daily recordings and additional 15 longer recordings ranging from 5 hours to multiple days.

We consider two types of advertising messages:

OS Advertising, i.e., regular BLE advertising activity for each considered OS (Section 5.1).

Accessory Activation, i.e., traffic originating from typical BLE-based accessories (Section 5.2).

Note that the sniffer also records advertising messages of other (non-tracked) devices in proximity, which are ignored.

5.1 OS Advertising

This scenario describes BLE advertising events which are periodically transmitted by an operating system when a device is in a Bluetooth-enabled state, without being triggered by a particular user action. In this scenario, the observer passively listens to a Bluetooth-enabled device while it is active for a certain amount of time. During our daily recordings over the period of four weeks, we identify whether advertising messages contain long-term static components.

We examine the following types of devices – all of which are owned and operated by the authors:

Windows 10 devices. We passively record advertising events, while the Bluetooth in Windows 10 is enabled. We measure data generated by one Microsoft Surface Pro 5, two laptops and one desktop PC from other OEMs, all running an up-to-date Windows 10.

macOS and iOS devices. We passively record advertising events, while Bluetooth in macOS / iOS is enabled. Additionally, the *Airdrop* sharing feature is launched on the respective devices, and the resulting advertising events are recorded. We measure data generated by various up-to-date iPhones

(iPhone 5s, iPhone 8, iPhone X, running iOS 11), two iPads (iPad, iPad Pro running iOS 12), and two Macs (iMac A1419 and Macbook Pro A1502 running macOS 10.13).

Android devices. We passively record advertising events, while Bluetooth in Android is enabled. We measure data generated by two different Android devices (Samsung Galaxy S5, Google Nexus 4).

Smartwatches. We connect each smartwatch to its respective app on a smartphone and then deactivate Bluetooth on the smartphone. This triggers the smartwatch to advertise its presence. We sample data from two Fitbit Charge 2 smartwatches intermittently over the course of multiple months. Additionally, one of the smartwatches is subjected to both a restart and a factory reset to find out if these actions affect its behavior.

5.2 Accessory Activation

This scenario describes advertising events triggered by the use of BLE-based accessories. Our work considers two types of touch-sensitive BLE-enabled pencils.

Microsoft Surface Pen. The Microsoft Surface Pro tablet can be used with a Microsoft Surface Pen. This digital pen is connected to the device via BLE in order to transmit pressure sensitivity data and furthermore transmit signals from its two buttons. Our experiment consists of disconnecting and reconnecting the pen in order to clarify whether the Surface Pro, or the pen use a static address or otherwise leaks privacy-relevant information. Furthermore, we examine the effect of button actuation for the same goal.

iPad Pro Pen. The iPad Pro can be used with the Apple Pencil pen. This pen is connected to the iPad via BLE in order to transmit pressure sensitivity data. Our experiment consists of disconnecting and reconnecting the pen in order to find out whether the iPad Pro, or the pen leak any privacy-relevant information.

OS	Prevalent Type	Prevalent Content
Windows 10	`ADV_NONCONN_IND`	Company Identifier `0x0006`
macOS, iOS	`ADV_IND`	Company Identifier `0x004c`
Android	`SCAN_REQ`	Target Address
Fitbit	`ADV_IND`	Fitbit Service Class UUID
Surface Pen	`ADV_DIRECT_IND`	Target Address

Table 2. Typical advertising features per operating system.

6 Results

6.1 Pre-Processing

In the pre-processing phase, we seek to identify suitable identifying tokens which may be used to track devices beyond their address randomization.

Operating system vendors implement widely differing strategies when it comes to implementing Bluetooth advertisements, driven by overall system architecture and ecosystem considerations. Therefore, different operating systems behave in different ways in order to announce their presence to nearby devices, and the OS origin is typically easily distinguishable by some high-level features in the broadcasted advertising messages.

Identifying the major OS vendors and device types is fairly straightforward, as Apple and Microsoft devices advertise using manufacturer-specific data, which is always prepended by an officially assigned Company ID [3]. In other cases where this approach is not used, the vendor can typically be found out by inspecting other specific characteristics like Service Class UUIDs [7, pp. 10, 12f., 18] or Service UUIDs [5].

The advertising message features shown in Table 2 allow for such a high-level classification of messages based on the considered devices.

6.1.1 Windows 10 devices

Windows 10 devices typically advertise a PDU Payload with only a single AD Structure of type *Manufacturer Specific Data*, as shown in Figure 6.

After recording for an extended period of time, we find the advertising address to regenerate approximately every 960 seconds, i.e., giving an adversary an average tracking time of approximately 16 minutes based on the address until the device is lost. This shows that Windows 10 implements *random private addresses* (see Section 3.3).

The Manufacturer Specific Data contained in the payload starts with the Company Identifier for Mi-

AdvA: <48bit random address>
AdvData: 1 AD Structure:
    0xFF → **Manufacturer Specific Data:**
        Company Identifier: 0x0006 → "Microsoft"
        Data: <27 bytes of data>

Fig. 6. A a typical Windows 10 advertisement PDU Payload, containing a single "Manufacturer Specific Data" AD Structure (see Figure 5).

crosoft [3] 0x0006 and 27 bytes of proprietary data (see Figure 6), comprised of four fixed bytes followed by 23 variable bytes[7]. The first fixed bytes do not differ between computers, i.e., they are not suitable to be used as identifying tokens. The 23 variable bytes, however, appear uniformly random and, in fact, seem to be unique per device.

Furthermore, the payload is observed to remain static for approximately one hour. We observed no differences in advertising behavior between the observed devices that would indicate manufacturer-specific, rather than OS-specific behavior. We therefore retain the following identifying token for Windows 10 devices:

$$A_{Win10} \quad = \quad \text{[Manufacturer Specific Data[-23:]]}. \quad (3)$$

6.1.2 Apple (macOS and iOS) Computers and Smartphones

MacOS and iOS devices regularly advertise their presence as well. The advertising rates in macOS and iOS are much higher than in Windows 10, at up to two advertising events per second. Address lifetimes are highly variable, ranging from seconds up to around two hours in rare edge cases, with an average of around 20 minutes.

The typical message format contains, similar to the Windows 10 payload, a manufacturer-specific data field. However, Apple devices can be further distinguished by advertising an additional *flags* data structure which Windows 10 does not use (see Figure 7). The manufacturer-specific data may contain one or more of Apple-specific data types, which are used for specific functionality in iOS and macOS which is facilitated by BLE. Table 3 shows an overview of the encountered data

AdvA: <48bit random address>
AdvData: 2 AD Structures:
    0x01 → **Flags:** 0x1a
    0xFF → **Manufacturer Specific Data:**
        Company Identifier: 0x004c → "Apple"
        Data: (**nearby** or **handoff** or both)
            nearby (0x10) <5 bytes of data>
            handoff (0x0c) <14 bytes of data>

Fig. 7. A a typical Apple device advertisement PDU Payload, containing a single "Manufacturer Specific Data" AD Structure (see Figure 5).

Description	iBeacon	AirDrop	Airpods	Airplay Source	Handoff	Nearby
Type	0x02	0x05	0x07	0x0a	0x0c	0x10
Occurrence	4,762	4,649	2,151	13,846	79,001	87,598

Table 3. Occurrences of the most common data types observed in iOS/macOS advertising messages.

types. Of the data types observed, the *nearby* and *handoff* types are by far the most frequent.

The *handoff* data structure consists of 14 bytes of data, in which the first bytes seems to be always zero and the remaining bytes may take arbitrary values. This data structure changes its value based on multiple factors, some of which we identify in Section 6.3.2. None of the observed values showed collisions between devices and the data has a sufficient length, making this a good candidate for an identifying token.

The *nearby* data structure consists of 5 bytes of data. Within these 5 bytes, the first two bytes do not seem to be fully random, reducing our useful length below the threshold set in Section 4.3.1, i.e., the number of bits is smaller than what we would want to avoid identity collisions. In practice, we have not observed values colliding, and have successfully been using *nearby* data in our experiments *as a second token* to the handoff data. We conclude that while it is of practical usefulness to our algorithm, it should not be relied on as a single identifying token.

Clearly, types of data which rarely appear in advertising messages are not suitable identifying tokens. We therefore consider the *nearby* and *handoff* data structures as the best candidates, since they appear an order of magnitude more often than any other data type (see Table 3).

An additional data type worth mentioning is the *Airdrop* data type. Airdrop allows users to share files with a nearby friend using an Apple-specific combina-

7 Example payload:
01092002 d70d650aac181b8397b8588f3a62de55810d23db56f176.
fixed bytes variable bytes

AdvA: <48bit random address>
AdvData: 3 AD Structures:
0x01 → **Flags:** 0x06
0x06 → **Incompl.List of 128-bit Service Class UUIDs:**
adabfb006e7d4601bda2bffaa68956ba
0x16 → **Service Data:**
uuid: 0x180a → "Device Information"
data: <5 bytes of data>

Fig. 8. A typical Fitbit PDU Payload (relevant AD Types are defined in [7, p. 10,p. 12f.,p. 18]).

tion of Wi-Fi and Bluetooth for peer-to-peer file transfer. Airdrop produces a significant amount of advertising events, which are triggered whenever a user taps or clicks a native share button in the respective OS. Interestingly, Airdrop uses a separate randomized advertising address for this purpose, which cannot be associated with the regular address used for other advertising – making it unsuitable as an identifying token.

Overall, we observed no differences in advertising behavior between the observed iOS or macOS devices that would indicate noteworthy behavioral differences between iOS 11, 12, or macOS 10.13. We therefore retain the identifying tokens

$$A_{Apple} = A_{iOS} = A_{macOS} \quad = \quad [\text{nearby, handoff}]. \quad (4)$$

6.1.3 Android smartphones

We observed Android advertising addresses to change in intervals of about 15-45 minutes. However, the observed Android smartphones use a completely different advertising approach than Windows or iOS/macOS, making them immune to the address-carryover algorithm. The tested Android phones never send out manufacturer-specific data or other potentially device-identifying data in regular intervals. Instead the OS scans for advertisements of other devices when the Bluetooth settings are opened by the user. Due to the lack of active, continuous advertising, identifying tokens cannot be assembled, making the observed Android devices immune to the carry-over algorithm.

6.1.4 Smartwatches

We analyze Fitbit smartwatches which use random addresses (according to their TxAdd flag) for advertising and provide a constant *128-bit Service Class UUID* as well as some *Device Information* data (see Figure 8). The smartwatch advertises approximately once every 7 seconds using this format.

While the TxAdd flag indicates that the devices use random addresses, the addresses actually never change (which has previously been analyzed in depth by Cyr et al. [12] and Das et al. [16]). In the tracking section, we provide some additional insights on this behavior, since such permanent trackability raises considerable privacy concerns.

The advertised 128-bit Service Class UUID `adabfb006e7d4601bda2bffaa68956ba` is specific to the watch model [30, 35], and is identical for the two recorded watches. Service Class UUIDs are therefore not suitable identifying tokens, as their value does not identify individual devices.

The Device Information data seems to be different for different smartwatches[8], and changes over time in the scale of weeks or months. We observed changes within one device which only concerned a single bit in the Device Information data, whereas differences between the observed devices differed in 7 different bits. More measurements on different devices would have to be conducted to conclude its fitness as an identifying token – given the above-mentioned trivial case of a never-changing address, we did not explore this further.

6.2 Device Tracking

Applying phase 2 of the algorithm reveal that multiple classes of devices are vulnerable to tracking beyond address randomization.

Windows 10 devices are vulnerable to the address-carryover algorithm because the advertising address and the identifying token A_{Win10}'s value changes occur at different intervals and out of sync. This allows the algorithm to either update the payload while the address is known, or vice versa. Figure 9 shows how we are able to track the device with the initial advertising address `1da6aa6d7b4e` and associate it with the device `2662e1021647` almost four hours later, and after 12 address randomizations. The longest observed MTT_{Win10} is 11.2 hours of continous tracking. However, assuming an always-on target device there is no clear upper bound, i.e., $MTT_{Win10} \in [11.2 \text{ hours}, \infty)$.

8 We did not have access to a sufficient amount of watches of this type to make a strong claim about its uniqueness

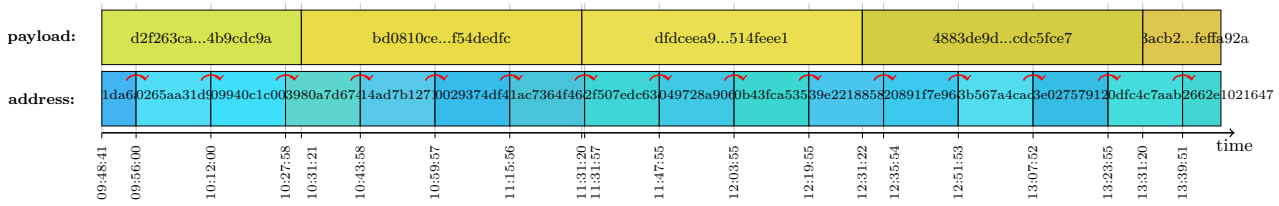

Fig. 9. An experiment illustrating the carry-over effect on Windows 10 devices. Asynchronous value changes allow updating the device identity whenever it changes.

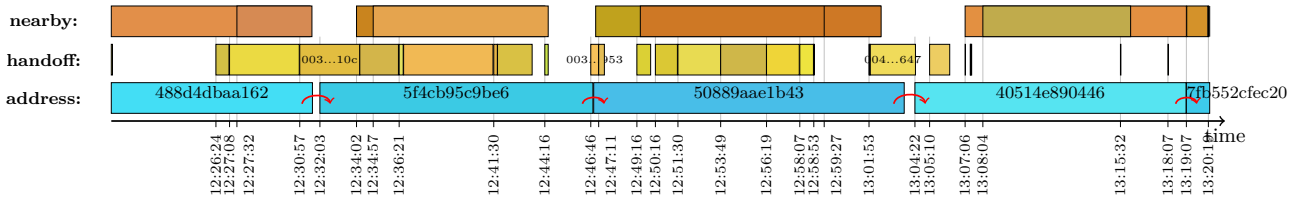

Fig. 10. This timeline shows address-carryover across 5 random addresses on iOS. The first three hops occur via the handoff identifying token, the last one occurs via the nearby token. Different colors denote different values.

iOS or macOS devices have two identifying tokens (nearby, handoff) which change in different intervals. In many cases, the values of the identifying tokens change in sync with the address. However, in some cases the token change does not happen in the same moment, which allows the carry-over algorithm to identify the next random address. Figure 10 shows such a carry-over chain, in which a device is trackable across five address randomizations. In contrast to the Windows 10 implementation, synchronized value changes appear to be more prevalent than asynchronous changes, the latter representing carry-over opportunities. $MTT_{Apple} = 53$ minutes is the longest time we were able to track during our experiments; additional tests may yield a longer time.

Prior findings by other groups [12, 16] indicate that **Fitbit** smartwatches advertising addresses do not periodically randomize. However, since the address carries the flag indicating a random address, we seek to further investigate its nature by subjecting the smartwatch to different conditions which may trigger a new random address. Our tests include:

- Draining the battery;
- Performing a hard reset;
- Performing a factory reset.

None of these measures resulted in a change of the observed advertising address, leading to permanent non-continuous trackability despite active attempts to regenerate the address. Hence, by confirming that there is no user-accessible way to anonymize the device identity periodicity, we conclude that there is no upper limit for the trackability of a Fitbit smartwatch of the type discussed, i.e., $MTT_{Fitbit} = \infty$.

Android devices do not appear to be vulnerable to our passive sniffing algorithm, as they typically do not send advertising messages containing suitable identifying tokens.

6.3 Other Vulnerabilities

This Section summarizes additional findings which do not directly result from the application of our algorithm, but were discovered in the process.

6.3.1 Accessory Activation

We look into the effect of BLE-connected accessories, specifically pencils of touch-enabled laptops and tablets.

The **Microsoft Surface Pen** has a button which can be configured to launch the note-taking (or any other) application to facilitate activities which make use of handwritten input. Whenever that button is pressed, the pen sends an `ADV_DIRECT_IND` request to its connected Surface device. These types of messages contain the device's own advertising address as well as the target address. In the case of the Surface Pen, this target address uses the *public static address* of the device it is

AdvA: <48bit random address>
AdvData: 2 AD Structures:
0x01 → **Flags:** 0x06
0x09 → **Complete Local Name:**
"Apple Pencil<13 Unicode Nullbytes>"

Fig. 11. Apple Pencil advertising during connection procedure..

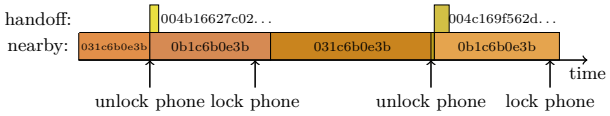

Fig. 12. The handoff data structure appears whenever an iOS device is unlocked.

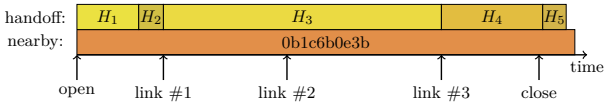

Fig. 13. The handoff payload changes correlate closely with certain user activity on iOS.

associated to. The Surface Pen also features a magnetic connector which allows it to be attached to the Surface. When the Surface Pen is detached from the Surface, it sends the same kind of address-leaking directed advertising as well.

This means that even if the Surface computer itself never discloses its public static address, the pen will expose it. The fact that it is a permanent identifier leads to the Surface computer being trackable permanently via active probing. Once the public address becomes known, an adversary could periodically probe the Surface device directly using its permanent address. The response to such a directed advertising message (or absence thereof) results in indefinite presence tracking of the Surface device, i.e., $MTT_{Surface,leaked} = \infty$.

We tested whether the **Apple Pencil** has similar issues, but it uses an entirely different approach. When it first connects to an iOS device, it emits a short burst of `ADV_IND` messages containing a random address, capability flags and a *Complete Local Name* field of value *Apple Pencil* (see Figure 11). After this initial advertising, the pen does not emit any advertising messages as long as it is paired with the iOS device. Every time the pen is set up again, the random address changes.

6.3.2 Activity Side-Channel on iOS

The handoff data structure only appears in certain patterns. Closer inspection reveals the following two characteristics:

- The handoff payload is triggered when a device gets activated;
- The handoff payload changes depending on context or content of the active application.

Both of these characteristics present a side channel which allows a passive observer to deduct activity patterns of the user using the target device. Figure 12 illustrates the presence of a burst of advertising messages containing the handoff data structure every time an iPhone is unlocked.

The handoff feature on macOS and iOS devices allows a user to continue one activity on another device. For example, it allows browsing a website on an iPhone and then continuing to browse it on a computer running macOS, when the two devices are in proximity. As such, this feature is inherently dependent on content and context of the user activity. It can be shown that handoff data changes highly correlate with user navigation using the Safari browser in iOS (see Figure 13).

7 Recommendations

Device implementations should follow a few simple rules in order to protect themselves against address-carryover tracking.

Synchronize payload changes with address randomizations. If the advertising message payload contains any type of data that could be used as an identifying token (see Section 4.3.1), the payload should change in sync with the address to prevent extended tracking. While there may be technical reasons for keeping certain payloads around longer than the default address randomization frequency of a given operating system, it should be ensured programmatically that there is no continuous carryover, as shown to be the case in Windows 10 and to a certain degree in iOS and macOS.

Implement address randomization for low-powered devices. For some devices, especially wearables and other battery-powered sensor devices, frequently randomizing the address may be

at conflict with energy constraints. In these situations, device states which are not concerned by these constraints should be leveraged to change the address, e.g., when charging the battery or when a power cycle or other maintenance activity is performed. For example, wearables typically have to be recharged weekly or even every few days, which – even though far from perfect – will at least prevent trivial address-based tracking for the lifetime of the device.

Implement reconnection addresses. BLE allows devices to exchange Identity Resolving Keys (IRK) which enable them to use resolvable random private addresses of each other (see Section 3.3). This allows for secure directed advertisement and connection initiation that does not leak permanent identifiers to the public [8, pp. 1251f., 2738]. Devices which currently use an advertising approach involving static addresses (such as the Microsoft Surface Pen) should consider integrating this protocol feature into their software architecture.

7.1 Workaround for Windows 10 devices

Our investigation showed that simply turning the Bluetooth setting in the *Windows 10 Settings Panel* off and on did not regenerate the payload and address used for Bluetooth advertising, instead merely paused advertising activity.

As Windows 10 is closed-source, a real fix has to be implemented by the manufacturer. However, it is still possible to break address-carryover tracking on the user side by completely disabling the Bluetooth device through the *Windows Device Manager* and re-enabling it again. Contrary to the Windows 10 Settings page, disabling the Bluetooth device in this manner *will* reset both the advertising address and the payload, thereby breaking the chain. Figure 14 shows an approach which practically achieves protection from our algorithm by cyclically performing this reset every x seconds whenever the device is not in the Connection state (in order to not break ongoing communication).

7.2 Workaround for iOS / macOS

For iOS and macOS, the same approach shown in Figure 14 applies. Switching Bluetooth off and on in the System Settings (or in the Menu Bar on macOS) will randomize the address and change the payload.

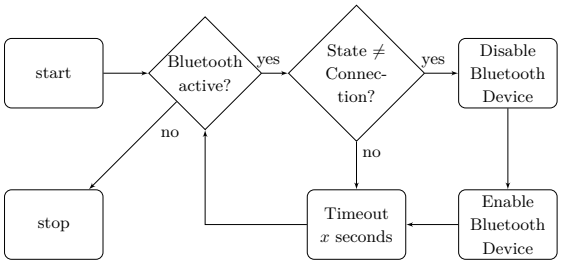

Fig. 14. A proposed workaround which breaks tracking via the address-carryover algorithm on Windows 10 and iOS/macOS devices.

8 Conclusion

Device manufacturers have flexibility in how to implement address randomization supported by BLE, and may be compelled to take shortcuts for various reasons – be it energy or memory constraints on the device level, software complexity or just for the sake of cutting development cost.

We showed that most computer and smartphone operating systems do implement address randomizations by default as a means to prevent long-term passive tracking, as permanent identifiers are not broadcasted.

However, we identified that devices running Windows 10, iOS or macOS regularly transmit advertising events containing custom data structures which are used to enable certain platform-specific interaction with other devices within BLE range. By observing typical advertising behaviors of these operating systems, we identified that parts of these data structures allow an adversary to abuse them as a temporary, secondary pseudo-identity. These *identifying tokens* can be integrated into an algorithm which allows device tracking beyond address randomization.

The **address-carryover algorithm** exploits the asynchronous nature of address and payload change, and uses unchanged identifying tokens in the payload to trace a new incoming random address back to a known device. In doing so, the address-carryover algorithm neutralizes the goal of anonymity in broadcasting channels intended by frequent address randomization.

The main findings of this work are summarized in Table 4. The algorithm succeeds consistently on Windows 10 and sometimes on Apple operating systems. In both cases, the respective identifying tokens change out of sync with the advertising address. In the Windows 10 case, there is no evidence of any synchronization by design. In the Apple case, it seems that there exist mecha-

Type	∅ Address life	MTT_*	Adversary	Mechanism
Windows 10	16 min	unbounded	continuous	carry-over algorithm
macOS, iOS	20 min	53 min[9]	continuous	carry-over algorithm
Android	15-45 min	no vuln. identified	continuous	carry-over algorithm
Fitbit Charge	static	unbounded	non-continuous	vendor implementation
Microsoft Surface	16 min	unbounded	non-continuous	identifier leak via Pen

Table 4. Trackability summary of the observed devices.

nisms to synchronize updates of identifying tokens with address randomization, but they occasionally fail.

The Android devices that we tested are not affected by the address-carryover algorithm, as they do not continuously send advertising messages. This is consistent with the BLE Central role (see Section 3.1), which scans for advertising from nearby Peripheral devices instead of advertising itself. Note that the Android SDK contains functionality for implementing BLE advertising [19], but it does not seem to be used by the OS by default in the devices we tested.

Fitbit has not fixed the trackability issue, despite it being known for years [12, 16]. This is concerning since such activity trackers are by design worn and carried around in daily life, and there is no way to prevent sniffing their permanent addresses.

Any device which regularly advertises data containing suitable advertising tokens will be vulnerable to the carry-over algorithm if it does not change all of its identifying tokens in sync with the advertising address. As Bluetooth adoption is projected to grow from 4.2 to 5.2 billion devices between 2019 and 2022 [9, p. 11], with over half a billion amongst them wearables and other data-focused connected devices [9, p. 15], establishing tracking-resistant methods, especially on unencrypted communication channels, is of paramount importance. This privacy concern is compounded by the realistic feasibility of BLE-based botnets [22] and complementary threats such as large-scale tracking of users via compromised Wi-Fi routers [31], which amplify trackability to a global scale. It can further be imagined that additional metadata such as electronic purchase transactions, facial recognition and other digital traces could be combined with Bluetooth tracking to generate a fine-grained location profile of a victim.

9 Longer tracking may be possible, but did not occur in our experiments.

8.1 Further Research

In this work, we entirely focused on the existence of data fields which could be re-purposed as pseudo-identifiers. The discoveries presented in this work do not require an SDR-based setup and should be reproducible using common Bluetooth communication hardware.

Future work may focus on additional factors such as the consideration of advertising message timing, which may make the address-carryover algorithm even more powerful. Another interesting factor would be physical layer device fingerprinting using SDR, similar to previous work done on other wireless technologies [2, 15, 36, 39], which may allow tracking *despite* immunity from the address-carryover algorithm to certain extents.

Furthermore, our current approach may be complemented by active attacks previously considered for Wi-Fi [22, 27, 34], and adapted to Bluetooth where applicable. This includes Connection-based attacks which are based on the data channels rather than just the BLE advertising channels.

Responsible Disclosure

Vulnerability to the address-carryover algorithm discovered in Microsoft and Apple software was disclosed with the companies in November 2018. Additional findings regarding the Microsoft Surface Pen and iOS activity side channel were subsequently disclosed to the respective companies during the ongoing correspondence.

Acknowledgment

This research was supported in part by NSF under grant CCF-1563753 and by the Boston University UROP program. The authors also thank Jed Crandall for shepherding the paper and the anonymous reviewers for their constructive suggestions.

References

[1] Apple Inc. iBeacon, 2014.

[2] Gianmarco Baldini, Raimondo Giuliani, Gary Steri, and Ricardo Neisse. Physical Layer Authentication of Internet of Things Wireless Devices Through Permutation and Dispersion Entropy. In *2017 Global Internet of Things Summit (GIoTS)*, pages 1–6. IEEE, 6 2017.

[3] Bluetooth Special Interest Group (SIG). Company Identifiers.

[4] Bluetooth Special Interest Group (SIG). Generic Access Profile.

[5] Bluetooth Special Interest Group (SIG). Service Discovery.

[6] Bluetooth Special Interest Group (SIG). *Bluetooth Core Specification. v4.0*. Bluetooth Special Interest Group (SIG), 2010.

[7] Bluetooth Special Interest Group (SIG). *Supplement to the Bluetooth Core Specification*. Bluetooth Special Interest Group (SIG), 2015.

[8] Bluetooth Special Interest Group (SIG). *Bluetooth Core Specification. v5.0*. Bluetooth Special Interest Group (SIG), 2016.

[9] Bluetooth Special Interest Group (SIG). Bluetooth Market Update 2018. Technical report, Bluetooth Special Interest Group (SIG), 2018.

[10] Bluetooth Special Interest Group (SIG). Core Specifications, 2018.

[11] Bluetooth Special Interest Group (SIG). Our History, 2018.

[12] Britt Cyr, Webb Horn, Daniela Miao, and Michael Specter. Security Analysis of Wearable Fitness Devices (Fitbit). *Massachusetts Institute of Technology*, pages 1–14, 2014.

[13] D.A. Dai Zovi and S.A. Macaulay. Attacking Automatic Wireless Network Selection. In *Proceedings from the Sixth Annual IEEE Systems, Man and Cybernetics (SMC) Information Assurance Workshop, 2005.*, volume 2005, pages 365–372. IEEE, 2005.

[14] Dino A Dai Zovi. KARMA Attacks Radioed Machines Automatically, 2005.

[15] Boris Danev, Davide Zanetti, and Srdjan Capkun. On Physical-Layer Identification of Wireless Devices. *ACM Computing Surveys*, 45(1):1–29, 2012.

[16] Aveek K. Das, Parth H. Pathak, Chen-Nee Chuah, and Prasant 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 - HotMobile '16*, pages 99–104, New York, New York, USA, 2016. ACM Press.

[17] Byron C Drachman and Michael J Cloud. *Inequalities: With Applications to Engineering*. Springer-Verlag, 1998.

[18] Google. Eddystone.

[19] Google Developers. BluetoothLeAdvertiser.

[20] Robin Heydon. An Introduction to Bluetooth Low Energy, 2013.

[21] IEEE Computer Society. *IEEE Standard for Local and Metropolitan Area Networks - Link Aggregation*. IEEE Standards Association, 2008.

[22] Taher Issoufaly and Pierre Ugo Tournoux. BLEB: Bluetooth Low Energy Botnet for large scale individual tracking. *2017 1st International Conference on Next Generation Computing Applications, NextComp 2017*, pages 115–120, 2017.

[23] Markus Jakobsson and Susanne Wetzel. Security Weaknesses in Bluetooth. In David Naccache, editor, *Topics in Cryptology — CT-RSA 2001*, pages 176–191, Berlin, Heidelberg, 2001. Springer.

[24] Mohamed Imran Jameel and Jeffrey Dungen. Low-Power Wireless Advertising Software Library for Distributed M2M and Contextual IoT. In *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, pages 597–602. IEEE, 12 2015.

[25] Xianjun Jiao. BTLE, 2014.

[26] Heikki Karvonen, Carlos Pomalaza-Ráez, Konstantin Mikhaylov, Matti Hämäläinen, and Jari Iinatti. Experimental Performance Evaluation of BLE 4 Versus BLE 5 in Indoors and Outdoors Scenarios. In Giancarlo Fortino and Zhelong Wang, editors, *Advances in Body Area Networks I*, pages 235–251. Springer, Cham, 2019.

[27] Jeremy Martin, Travis Mayberry, Collin Donahue, Lucas Foppe, Lamont Brown, Chadwick Riggins, Erik C. Rye, and Dane Brown. A Study of MAC Address Randomization in Mobile Devices and When it Fails. *Proceedings on Privacy Enhancing Technologies*, 2017(4):365–383, 10 2017.

[28] Radius Networks. AltBeacon, 2015.

[29] reelyActive. reelyActive-git.

[30] reelyActive. Sniffypedia, 2018.

[31] Pierre Rouveyrol, Patrice Raveneau, and Mathieu Cunche. Large Scale Wi-Fi Tracking Using a Botnet of Wireless Routers. *Workshop on Surveillance & Technology*, 2015.

[32] Krishna Sampigethaya, Leping Huang, Mingyan Li, Radha Poovendran, Kanta Matsuura, and Kaoru Sezaki. CARAVAN: Providing Location Privacy for VANET. Technical report, Washington Univ Seattle Dept of Electrical Engineering, 2005.

[33] Dominic Spill and Andrea Bittau. BlueSniff: Eve meets Alice and Bluetooth. *WOOT '07 Proceedings of the first USENIX workshop on Offensive Technologies*, page 10, 2007.

[34] Mathy Vanhoef, Célestin Matte, Mathieu Cunche, Leonardo S. Cardoso, and Frank Piessens. Why MAC Address Randomization is not Enough. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security - ASIA CCS '16*, pages 413–424, New York, New York, USA, 2016. ACM Press.

[35] virtualabs. probeZero, 2016.

[36] Tien Dang Vo-Huu, Triet Dang Vo-Huu, and Guevara Noubir. Fingerprinting Wi-Fi Devices Using Software Defined Radios. In *Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks - WiSec '16*, pages 3–14, New York, New York, USA, 2016. ACM Press.

[37] Isabel Wagner and David Eckhoff. Technical Privacy Metrics. *ACM Computing Surveys*, 51(3):1–38, 2018.

[38] Martin Woolley. Bluetooth Technology Protecting Your Privacy, 2015.

[39] Qiang Xu, Rong Zheng, Walid Saad, and Zhu Han. Device Fingerprinting in Wireless Networks: Challenges and Opportunities. *IEEE Communications Surveys & Tutorials*, 18(1):94–104, 2016.