Every Byte Matters: Traffic Analysis of Bluetooth Wearable Devices

LUDOVIC BARMAN, EPFL
ALEXANDRE DUMUR, EPFL
APOSTOLOS PYRGELIS, EPFL
JEAN-PIERRE HUBAUX, EPFL

Wearable devices such as smartwatches, fitness trackers, and blood-pressure monitors process, store, and communicate sensitive and personal information related to the health, life-style, habits and interests of the wearer. This data is typically synchronized with a companion app running on a smartphone over a Bluetooth (Classic or Low Energy) connection. In this work, we investigate what can be inferred from the metadata (such as the packet timings and sizes) of encrypted Bluetooth communications between a wearable device and its connected smartphone. We show that a passive eavesdropper can use *traffic-analysis attacks* to accurately recognize (a) communicating devices, even without having access to the MAC address, (b) human actions (e.g., monitoring heart rate, exercising) performed on wearable devices ranging from fitness trackers to smartwatches, (c) the mere opening of specific applications on a Wear OS smartwatch (e.g., the opening of a medical app, which can immediately reveal a condition of the wearer), (d) fine-grained actions (e.g., recording an insulin injection) within a specific application that helps diabetic users to monitor their condition, and (e) the profile and habits of the wearer by continuously monitoring her traffic over an extended period. We run traffic-analysis attacks by collecting a dataset of Bluetooth communications concerning a diverse set of wearable devices, by designing features based on packet sizes and timings, and by using machine learning to classify the encrypted traffic to actions performed by the wearer. Then, we explore standard defense strategies against traffic-analysis attacks such as padding, delaying packets, or injecting dummy traffic. We show that these defenses do not provide sufficient protection against our attacks and introduce significant costs. Overall, our research highlights the need to rethink how applications exchange sensitive information over Bluetooth, to minimize unnecessary data exchanges, and to research and design new defenses against traffic-analysis tailored to the wearable setting.

CCS Concepts: • **Security and privacy → Mobile and wireless security**.

Additional Key Words and Phrases: traffic-analysis,metadata,Bluetooth,wearables

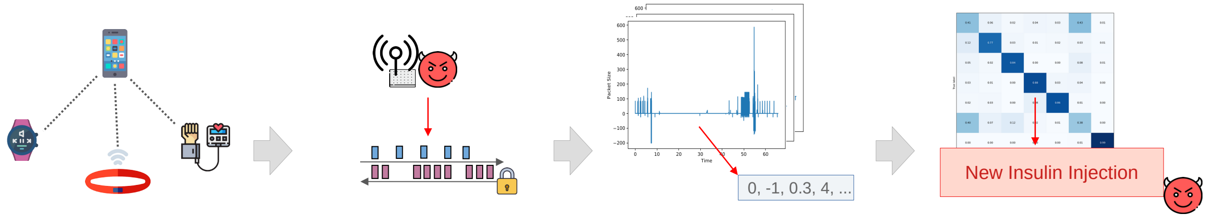
ACM Reference Format:
Ludovic Barman, Alexandre Dumur, Apostolos Pyrgelis, and Jean-Pierre Hubaux. 2021. Every Byte Matters: Traffic Analysis of Bluetooth Wearable Devices. In ,. ACM, New York, NY, USA, 46 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

With the rising interest in personalized health care and “quantified self", wearable Bluetooth devices are becoming pervasive in our societies. To improve aspects of their daily lives, users increasingly[1] use smartwatches, medical and fitness-related devices such as activity trackers, step counters, blood-pressure monitors and sleep trackers. These devices process, store, and transmit personal and sensitive data linked to the wearer’s identity and health status: fine-grained and long-term activity levels, heart rates, arrhythmia alerts, medication and sleep schedules,

[1] In 2019, 46 million wearable devices were sold. This number is expected to rise to 158 millions by 2022 [10].

IMWUT ’21, Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.
© 2021 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn



(1) Wearable devices communicate with a smartphone using Bluetooth

(2) An adversary eavesdrops upon their encrypted communications

(3) Feature extraction

(4) Traffic classification: identification of sensitive actions

Fig. 1. Traffic-analysis attack on the encrypted traffic of Bluetooth wearable devices. After an offline training phase (not depicted here), the passive eavesdropper can recognize the action to record an insulin injection despite the use of encryption.

etc. Such personal information should be protected from third parties, as it can be used to build profiles, to identify and track users (e.g., for advertising purposes), or can be sold to insurance companies for the quantification of users' medical risks. Medical wearable devices that are FDA-approved[2] are already subject to such a requirement: they "should have appropriate protections in place that prevent sensitive information from being read by unauthorized parties either in storage or in transmission" [3].

For enhanced functionalities, most wearable devices communicate with a smartphone via Bluetooth. These devices can use encryption at the Bluetooth link-layer or application-layer. In this work, we show that these encrypted communications leak information about their contents via their communication patterns (e.g., distribution of packet sizes and inter-arrival timings). We consider a local passive eavesdropper who attempts to infer sensitive information from the communications. This adversary could be a nosy neighbor, an advertiser in a shopping mall attempting to infer the shoppers' habits, an employer [13], or a more nefarious adversary collecting data for sale to insurance companies. These threats against privacy are concrete: For over a decade, advertisers have been using Bluetooth, Wi-Fi, and cellular networks, to track consumers in stores [4, 5, 7], and to link their profiles to online advertisement databases [11]; these advertisers could use Bluetooth traffic patterns to learn new information or to better profile users.

We infer sensitive information from the encrypted communications of wearable devices by using *traffic-analysis attacks* [44], a technique that exploits the communication patterns (e.g., packet sizes and timings) of encrypted traffic. These attacks have been successfully demonstrated in diverse settings: to recognize web pages on Tor traffic [48, 67, 88, 89], to fingerprint devices [68] or to infer the activities performed in a user's smart home [17, 81] and to recognize user activities and applications used on a smartphone (e.g., sending an e-mail or browsing a web page) [42, 76, 83, 84, 94]. The focus of recent related works has been on inferring user activities in the IoT and smart home setting [17, 21, 27, 28, 85], eavesdropping on WLAN or Internet traffic (or both). Very few related works consider the communications of Bluetooth wearable devices at the Personal Area Network (PAN) scale, despite the sensitive nature of the information exchanged. Das *et al.* [45] demonstrate how the bitrate of some fitness trackers is correlated with the user's gait; however, their analysis is restricted to 6 BLE devices, and they do not attempt to recognize devices, applications or users actions. To the best of our knowledge, ours is the first work that performs in-depth device, application and user-action identification on the Bluetooth communications of wearable devices.

To perform the traffic analysis of wearable devices, we build a Bluetooth (Classic and Low Energy) traffic-collection framework. We set up a testbed consisting of a diverse set of wearable devices (smartwatches, fitness trackers, blood-pressure monitors, etc.) connected to a smartphone, and we use a Bluetooth sniffer to capture the traffic. To generate realistic traffic traces, we manually use the wearable devices in the intended way: e.g.,

[2] U.S. Food and Drug Administration. This institution notably edicts rules for medical devices sold in the U.S.

in the case of a fitness monitor, by repeatedly performing a short running exercise until we obtain a sufficient number of samples. We obtain a dataset of labeled (encrypted) Bluetooth Classic and Low Energy traces; we then design features based on packet sizes and timings, and we apply machine-learning classification techniques for identifying devices, applications, and user actions despite the encryption.

Our experimental results show that an eavesdropper can exploit encrypted communication patterns to passively identify devices, even in the absence of Bluetooth addresses or friendly names. We show that the timings of encrypted communications allow to identify specific models/versions of a device, and hence defeats the Bluetooth address-randomization protection employed by the Bluetooth Low Energy protocol. Moreover, we demonstrate that the adversary, for instance a nosy neighbor, can use our traffic-analysis attacks to accurately recognize user actions (e.g., recording the heart rate, beginning a workout or receiving an SMS) performed on different wearable devices such as smartwatches, step counters, and fitness trackers. We then focus on smartwatches and show that an eavesdropper, for instance an unscrupulous advertiser, can recognize the mere opening of specific applications, which can be immediately sensitive (e.g., in the case of a medical app) or used to build a profile (e.g., based on religious or political apps). Furthermore, we show that our methodology generalizes well: The model trained on a smartphone/wearable device pair performs accurately on a different device pair, indicating its cost-effectiveness. We also highlight how a targeted adversary can recognize sensitive user activities within a specific application; we accurately recognize the action of recording an insulin injection in a diabetes-helper application (Figure 1). Finally, we show that an attacker can infer a wearer's habits and build her profile when eavesdropping her Bluetooth traffic over a long time-period.

Finally, we focus on countermeasures against the presented traffic-analysis attacks. We evaluate how standard approaches against traffic analysis (i.e., padding or delaying messages, injecting dummy traffic) perform against our attacks. Our results show that these defenses provide insufficient protection: though they yield a drop in the adversary's accuracy, none of them reduces it to that of random guessing. Furthermore, their costs are high (from tens to hundreds of kilobytes of additional exchanged data) for wearable device communications where energy consumption is crucial. We also observe that the effectiveness of these defenses varies greatly with the adversarial task: although the "right" defense drops the attacker's accuracy, non-adapted defenses have almost no effect yet still incur high costs. This suggests that a "one-size fits all" defense for preventing device fingerprinting, as well as application and action fingerprinting, is unlikely to exist at a reasonable cost. Our defense evaluation highlights the need to rethink how wearable devices exchange sensitive data over Bluetooth communications and prompts for the design of novel defenses. For example, for applications that can support it, *data minimization* is a valid strategy: data that is not exchanged cannot be fingerprinted, and we observe in our experiments that low-volume exchanges are naturally protected from traffic analysis. Moreover, bulk transfers (e.g., synchronizing a step counter every day at midnight) also hamper the task of the adversary.

Overall, the purpose of our work is to raise awareness, notably among device manufacturers and application developers, of the limited confidentiality on the Bluetooth link with today's applications and devices. For instance, manufacturers might be willing to implement mitigations directly to the wearable devices' firmware, hence it is urgent that the research community provides them with acceptable solutions that will protect the next generation of wearable devices. Application developers might want to carefully consider the information their application can leak through its communication patterns and to implement an application-level defense (for instance, pad all communications to a fixed size). We hope that our research results will be a starting point for further research on the communication metadata of Bluetooth wearable devices. To facilitate future research on the topic, we open-source our dataset for research purposes: it consists of Bluetooth traces of wearable devices used to perform multiple actions and that have been, for most devices, manually recorded over months. Although our experiments focus only on Bluetooth communication metadata, our dataset contains all captured data (e.g., link-layer information, pairings) that might be of independent interest.

In summary, the contributions of this work are as follows:

- We show traffic-analysis attacks that, based on the encrypted traffic of wearable devices, recognize:
  1. communicating devices, up to the model number of the same device;
  2. user activities and the opening of specific applications;
  3. fine-grained sensitive actions (i.e., recording an insulin injection) within an application;
  4. actions recorded over a long time-period, which can be used to build a profile.
- We experimentally evaluate standard defenses against traffic analysis and suggest possible strategies;
- We make available a large dataset of Bluetooth traffic captures for future research.

The remainder of the paper is organized as follows: in §2, we introduce background information about the Bluetooth protocol used by wearable devices for their communications. In §3, we describe the system and threat model considered in our work, and in §4, we present the methodology employed for our dataset collection and the traffic-analysis attacks. Then, in §5 and §6, we demonstrate the results of our traffic-analysis attacks that identify devices, actions and applications, from the encrypted Bluetooth traffic of wearable devices. In §7, we evaluate the performance of standard defenses against our attacks. We discuss the contributions of the paper and its limitations in §8 and §9, the related work in §10, and then conclude in §11.

2 BLUETOOTH BACKGROUND

There exist two flavors of Bluetooth: Bluetooth Classic, referred to as "BR/EDR" for Basic Rate/Enhanced Data Rate, and Bluetooth Low Energy (BLE). The former is used for data-intensive or latency-sensitive scenarios (e.g., audio streaming), whereas the latter is used for low-power or low-throughput scenarios. Most wearable devices we collected for our testbed use Bluetooth Low Energy, except for smartwatches that typically use Bluetooth Classic. Both Bluetooth specifications [32] are produced by the Bluetooth Special Interest Group (SIG), and their latest version is 5.2.

Data Exchange. In both Bluetooth Classic and BLE, data exchange occurs after a pairing process between one or more slaves (the wearable device) and a master (e.g., the smartphone). We remark that a smartwatch can be a master for another wearable device. Initially, to discover each other, devices broadcast and listen on *advertising channels* in Bluetooth Low-Energy, or on channels determined by a predefined hopping sequence in Bluetooth Classic. Then, the pairing process assigns the slave to the master's *Piconet*, and both devices communicate using the non-advertising channels. Unlike Bluetooth Classic, BLE also supports data exchange without establishing a link-layer connection: devices simply broadcast short information to its surroundings on the advertising channel.

To communicate in a Piconet, all devices use the same frequency hopping pattern (derived from the master's MAC address and clock). The channel is divided into time-slots; odd time-slots are for the slaves, and even time-slots for the master. When the connection is Asynchronous Connectionless (ACL), devices communicate opportunistically during unreserved time-slots. In the case of Synchronous Connection-Oriented (SCO) connections, devices communicate at predetermined time-slots without acknowledgment.

Security. The security properties are similar between Bluetooth Classic and BLE. Devices first establish and authenticate a long-term key through a *pairing* process. In this process, both devices also derive a short- or long-term key. "Legacy" pairings are generally insecure by today's standards [91]: they rely on a short PIN and can be easily brute-forced. Secure Simple Pairing (SSP) is a more recent pairing protocol based on elliptic curve cryptography [70]. It is available since Bluetooth 2.1.

To authenticate the long-term key, four authorization mechanisms exist:

- JustWorks, which uses a hardcoded key;
- Out-of-Band, which relies on an external channel (such as NFC);
- Numeric Comparison, where the user visually compares numbers;
- Passkey Entry, where the user inputs twice the same code on the devices.

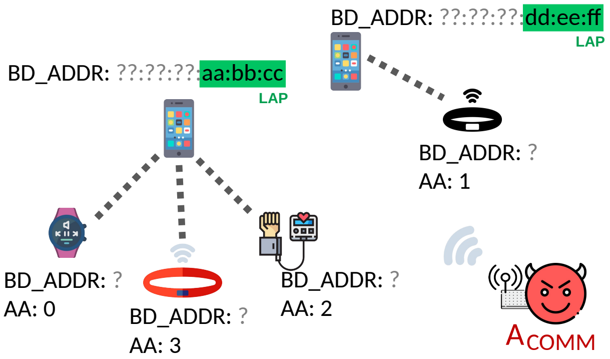


Fig. 2. Information visible to A_{comm}: (1) the active connections between each pair of wearable-smartphone, identified by a random Access Address (AA); A sees all packets on these connections; (2) for each master device, the Lower Address Part (LAP) of the BD_ADDR (sometimes called the MAC address). A_{comm} cannot tell what types of devices are communicating.

There exists several active attacks on SSP, based on some form of Man-in-the-Middle [50–52] or low-entropy key generation [25, 26]. If the pairing is done correctly and the key uses sufficient entropy, the subsequent communication should be confidential and integrity-protected (AES-CCM with a 128-bit key).

Bluetooth Eavesdropping. Eavesdropping on Bluetooth traffic is complex due to the frequency hopping. Inexpensive sniffers (such as the Ubertooth [16]) attempt to follow the frequency hopping of a connection and often require observing the pairing [75]. However, by listening concurrently on all channels, high-end commercial scanners accurately capture all traffic without the need to observe the pairing. Recent advances in research use coordinated Ubertooth devices to achieve a greater capture accuracy [19, 20], or use software-defined radios (SDRs) to cover the Bluetooth spectrum at a cost lower than commercial scanners [40, 41]. We discuss the practicality of Bluetooth eavesdropping in §8.

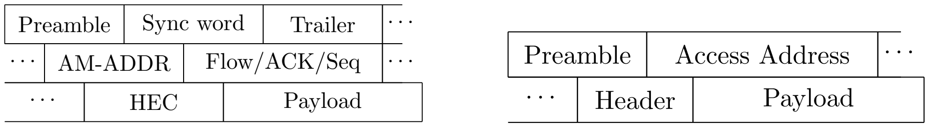
3 SYSTEM AND ADVERSARIAL MODEL

We consider a user U that possesses a smartphone S and a collection of wearable devices W. This collection W contains heterogeneous devices: some with a general-purpose OS (e.g., an Android smartwatch) or a simpler firmware (e.g., a step counter). Devices in W can communicate with S over Bluetooth Classic or Low Energy.

Adversary. We consider an adversary A who is a passive eavesdropper. A uses a Bluetooth sniffer to capture all Bluetooth traffic in the vicinity of U. In particular, A might capture traffic from other users and devices that are also nearby. A does not compromise any devices and does not possess the keys to decrypt Bluetooth traffic.

Goals and Traffic Captured. Informally, A attempts to infer information from all communications between the wearable devices in W and S (Figure 2). However, we need to precisely define the aforementioned adversary. In practice, an eavesdropper who sees *all* traffic between a wearable device and its connected smartphone is not realistic: a sporadic or local adversary is unlikely to consistently capture one-time events such as pairings. Therefore, we define as A_{all} an adversary who sees all communications between a wearable-smartphone device pair, and we further define two weaker adversaries in the sense of A, but who only observe, respectively:

- A_{paging}: all paging events (devices waking up from sleep and performing minimal discovery) and subsequent communications, but not the pairings;
- A_{comm}: all active communications, but neither pairing nor paging events.



(a) Structure of a BB_PDU packet (BaseBand Packet Data Unit) used in Bluetooth Classic. HEC (Header Error Correction) is an error-correction code.

(b) Structure of a Link Layer packet used in Bluetooth LE.

Fig. 3. Packet Structures. The features used in our attacks derived from the time of reception and the size of the payload.

In terms of adversarial power, we have that $A_{\text{all}} \geq A_{\text{paging}} \geq A_{\text{comm}}$. We focus on the weakest adversary A_{comm} who only observes ongoing communications between a wearable device and its connected smartphone. This matches what a passive adversary can do consistently.

Information Visible to A_{comm}. Figure 3(a) shows the information that is visible to A_{comm} during Bluetooth Classic communications. The sensitive/identifying fields are the Sync word and the Active-Member Address (AM-ADDR). The Sync word reveals the 24 lower bits of the master's MAC address (LAP). The Upper Address Part (UAP), that is not transmitted in this packet, is assigned to manufacturers and identifies a wearable device. Finally, the Active-Member Address is a 3-bit integer (and not a MAC address) identifying a device in a given Piconet; this is similar to a connection identifier. Figure 3(b) shows the information visible to A_{comm} for Bluetooth LE. Similar to AM-ADDR, the Access Address is a connection identifier (and not a MAC address). On a higher-level, in both Bluetooth flavors, the information given to A_{comm} per packet is

$$(\text{Packet type},\ \text{connection ephemeral ID},\ \text{time},\ \text{payload})$$

where the Packet type indicates whether Bluetooth Classic or LE is used, and where the connection ephemeral ID is randomized, except for the master device in Bluetooth Classic where it is fixed. The payload can contain upper-layer packet information, for instance the packet type, as well as the encrypted application payload.

Capabilities of A_{comm}. In conclusion, with the information visible on the Bluetooth channel, A_{comm} can

- enumerate the devices $w \in W$ and assign a pseudonym to each of them;
- group them by connection between a pair of devices; and
- collect a series $\{(\text{time},\ \text{packet type},\ \text{packet size})\}_w$ for each device $w \in W$ for the duration of the capture.

We assume that the adversary is able to isolate a communication of interest, e.g., using the Active-Member Address (AM-ADDR) of Bluetooth Classic or Access Address of Bluetooth Low Energy, possibly combined with other techniques, e.g., distance estimation using the Received Signal Strength Indicator (RSSI). The pseudonymous recipient of a packet is often known, as most packets are acknowledged.

Attacker Goals. Given the Bluetooth information available, the goal of A_{comm} is threefold:

G1) **Device Identification:** for a device w, recognize the device brand/manufacturer;
G2) **Action Identification:** for a device w, recognize user actions (i.e., interactions with the device);
G3) **Application Identification:** for a device w, recognize the running application.

We use Goal 1 as a stepping-stone for the following goals. However, meeting the first goal already has privacy implications: such an adversary can recognize and track a user through the list of her wearable devices, despite MAC address randomization, and can build a profile for that particular user.

4 METHODOLOGY & DATASET

We describe how the adversary (defined in §3) eavesdropping on Bluetooth communications can perform traffic-analysis attacks to achieve its goals. We present the methodology that applies to the traffic-analysis attacks presented in §5 and §6. We first build a data collection framework to record Bluetooth traffic and to automate the data transmission for some Bluetooth wearable devices. For other devices, we manually trigger Bluetooth traffic by performing the appropriate human action (e.g., pressing a sequence of buttons, performing a physical activity). Then, we process the captured traffic and use it with machine-learning classification techniques to infer information from encrypted Bluetooth traffic.

Testbed. We set up a testbed with multiple wearable devices to account for a wide range of manufacturers, device capabilities, and functionalities. We initially selected 18 Bluetooth Classic and LE devices: 5 popular smartwatches, 2 headphones, and 11 other wearable devices (step counters, fitness trackers, and blood-pressure monitors) from the most popular vendors.

All Bluetooth Classic devices analyzed use link-layer encryption. To our surprise, from the Bluetooth Low Energy devices, only the Fitbit Charge 2 uses link-layer encryption. We perform entropy tests to validate that the remaining Low Energy devices use encryption at the application layer. We observe a high entropy ($6-8$bit of information per byte) for all devices, except for 3 of them (the `Beurer AS80`, `SW170`, and `PanoBike+`, ≈ 4bit/byte). Hence, we discard these three devices from our testbed. We remark that though our attacks do not use plaintext contents and would work on these three devices, an attacker can achieve the same results by simply reading the payload. We also discard two blood-pressure monitors (the `Qardio` and `H2-BP`) that we could not reliably use. In total, the testbed consists of 13 devices: 7 Bluetooth Classic and 7 Low Energy devices (Tables 1 and 2).

Although our testbed consists of a modest number of devices, most devices use proprietary firmware and software (i.e., they are closed-source), which does not allow us to automate their Bluetooth traffic-data collection. Except for Wear OS smartwatches that we can automate, the data collection is a sequential and manual process.

We use a Nexus 5 as the smartphone for all Android/Tizen smartwatches and wearable devices without an OS, and an iPhone 8 for the Apple Watch/AirPods. We updated all devices with the latest firmware and OS updates.

Actions Recorded. For each device, we manually compile a set of possible user-actions: low-end devices sometimes only `Sync` with the smartphone; mid-range devices support activities such as `Running Workout`, `Measure Heartbeat`, etc., whereas high-end devices such as smartwatches offer a large range of user actions through the use of additional apps (e.g., `AppX Open`, `AppX DoActionY`) that users can choose to personalize their device.

Capture Methodology. We record all Bluetooth traffic by using a wide-band scanner (an Ellisys Vanguard [9]). For each device w, we pair w with the phone S and let some time pass to allow for an initial data synchronization. Then, we trigger the desired action and record the corresponding Bluetooth traffic for $T = 30$ seconds (we observe this to be a conservative value); this constitutes one sample. To collect sufficient number of samples for an action, we repeat the capture process $N = 25$ times (we observe this to be a conservative value, Figure 11(c)). Each sample corresponds to a distinct recording; hence, samples are independent from each other. We manually select the device of interest and discard traces from other devices.

We record real activities: for instance, for a `Running Workout`, we perform a short running activity in the vicinity of the sniffer; for the wearable that records the heart rate, we fit the wearable to ourselves during the experiment. This is in contrast with some previous works (in the context of Wi-Fi traffic) that use UI fuzzing to quickly provide many traffic samples generated by smartphones [42, 83, 84]. This ensures that most values communicated by the wearable reflect what a real user would generate (e.g., the number of steps, speed, and distance). However, due to our non-portable experimental setup, the recorded GPS coordinates will show an almost-fixed position.

Environment. We conduct the experiments in an office where both Wi-Fi devices and other Bluetooth devices communicate. This corresponds to a noisy environment; we note that an anechoic chamber or a Faraday cage would advantage the adversary by allowing to record trace with less noise.

Dataset. We collect a dataset that consists of 10,700 Bluetooth captures with a duration of $\approx$ 30sec, recorded over 13 devices, 80 applications and 32 user-actions (see Table 3). This amounts to $\approx$ 98 hours of recording and represents 21GB of data. 10,371 of these traces are over Bluetooth Classic, and 329 Bluetooth Low Energy. We note that 2,215 of these captures are the result of the corresponding human action (e.g., performing a short workout), and 8,485 are automated actions (e.g., the opening of an application) on Wear OS devices.

Additionally, we record a second dataset that contains longer captures with a duration of $\approx$ 20min. These captures contain automated actions that, when combined, represent a plausible usage of a smartwatch over a day. We use them to model a persistent adversary (§6.2.4). This amounts to 38 hours of recording and 9.5GB of data.

Overall, the datasets contain raw binary files with all captured traffic. As the file format is proprietary, we extract the corresponding CSV files. We make the datasets (and all code used in this paper) available for research purposes [30].

Sample Processing. We process the captured raw Bluetooth traces as follows. We follow the approach of the related work and extract only the total length of the payload to avoid relying on the presence of plaintext markers in it [83, 84]. From each recorded Bluetooth trace, we extract the Bluetooth packets at the Logical Link Control and Adaptation Protocol (L2CAP) layer and output a series of (arrival time, size)-tuples. We label the trace with the `device` used, the `application` used (in the case of a smartwatch), the `action` performed, and whether it used Bluetooth Classic or Low Energy. We split the set of recorded samples S into two subsets S_{Classic} and S_{LE} depending on the Bluetooth variant used. We remark that an adversary is able to do the same, as the frames are different.

Feature Extraction. We design features to capture patterns based on incoming/outgoing size distributions and inter-packet timings. We map each sample in S_{Classic} and S_{LE} to a 32-scalar feature vector: First, we capture global statistics about packet sizes. We filter each sample into 3 packet sequences: (1) from Master to Slave, (2) from Slave to Master, and (3) sequences consisting of all non-null, non-ACK packets. From each filtered sequence, we extract the following 5 statistics: the min/mean/max/count/standard deviation of the packet sizes in bytes. Then, we observe that different devices/applications send packets with a distinctive size. To this end, following the techniques used by Liberatore and Levine [60] and Herrmann *et al.* [54] in the context of website fingerprinting, we extract features corresponding to the number of packets that are in a certain range. We select 10 buckets that represent the size (in bytes) ranges $[0, 9], [10, 19], \cdots, [80, 89]$ and a last "catch-all" bucket for packets with $[90, \infty]$ bytes. Finally, we examine the timings of the packets. In more detail, we compute, in seconds, the series of inter-packet durations and extract the same 5 statistics (min/mean/max/count/standard deviation) from it. Furthermore, we calculate the average send/receive inter-packet times as done by Saltaformaggio *et al.* [76] in the case of TLS traffic: $\text{AvgIPT}(P) = \frac{\sum_{i=0}^{|P|-1} \text{ts}_{i+1} - \text{ts}_i}{|P|-1}$, where P is the set of sent/received packets, and ts_i is the timestamp of packet i. Intuitively, this captures the communication bursts of each device.

Classification. For our traffic-analysis attacks (described in §5 and §6), we use a Random Forest classifier, a popular algorithm for multi-class classification that is also widely used in the related work [83, 84]. We choose a Random Forest classifier over the more recent deep-learning approaches [78, 79] due to its interpretability and the moderate size of our dataset. We split the samples of each device/application/action (depending on the adversarial

Bluetooth wearable devices used in our experiments. BT indicates the Bluetooth version. The `AppleWatch` uses both flavors of Bluetooth.

Table 1. Bluetooth Classic devices.

Vendor	**Model**	**OS**	**BT**	**Chipset**
Samsung	Galaxy Watch	Tizen OS	5.0	Broadcom
Fossil	Explorist HR	Wear OS	4.2	Qualcomm
Apple	Watch 4	Watch OS 5	5.0	Apple
Huawei	Watch 2	Wear OS 2	4.1	Broadcom
Fitbit	Versa 2	Fitbit OS 4	5.0	Cypress S.
Sony	MDR-XB9	—	4.1	Qualcomm
Apple	AirPods 2	—	5.0	Apple

Table 2. Bluetooth Low Energy devices.

Vendor	**Model**	**BT**	**Chipset**
Apple	Watch 4	5.0	Apple
Fitbit	Charge 2	4.1	Microelectronics
Fitbit	Charge 3	5.0	Cypress Semiconductor
Huawei	Band 3e	4.2	RivieraWaves
Mi	Band 2	4.1	Dialog Semiconductor
Mi	Band 3	4.1	Dialog Semiconductor
Mi	Band 4	5.0	Dialog Semiconductor

Table 3. Applications and Actions captured in the dataset. Some actions are application-specific, e.g., `DiabetesM_AddCalorie`. Other actions, e.g., `Workout`, exist in different apps and in the firmware of wearable devices.

Applications	20Min, ASB, Alarm, AppInTheAir, AthanPro, AthkarOfPrayer, Battery, BeurerApp, Bild, Bring, Calm, Camera, ChinaDaily, Citymapper, DCLMRadio, DailyTracking, DenverApp, DiabetesM, DuaKhatqmAlQuran, Endomondo, FITIVApp, FITIVPlus, FindMyPhone, Fit, FitBreathe, FitWorkout, Fitbit, Flashlight, FoursquareCityGuide, Glide, GooglePay, GooglePlayMusic, HealthyRecipes, HeartRate, HuaweiApp, Kaia, KeepNotes, Krone, Lifesum, MapMyFitness, MapMyRun, Maps, Medisafe, Meduza, MiApp, Mobills, Music, MyFitnessPalApp, NYT, NoApp, Outlook, PearApp, Phone, PhotoApp, PillReminder, PlayMusic, PlayStore, Qardio, RamadanTime, Reminders, Running, SalatTime, SamsungHealthApp, Shazam, Sleep, SleepTracking, SmartZmanim, SmokingLog, Spotify, Strava, Telegram, Timer, Translate, Walgreens, WashPost, WearCasts, Weather, Workout.
Actions	AddCalorie, AddCarbs, AddFat, AddFood, AddGlucose, AddInsulin, AddProteins, AddWater, Browse, BrowseMap, CaloriesAdd, Coffees, EmailReceived, HeartRate, Leisure, LiveStream, NightLife, Open, PhoneCallMissed, PhotoTransfer, Play, Restaurants, Running, SearchRecipe, Shopping, Skip, SMSReceived, Sync, Walking, Workout.

goal) into 80% training and 20% testing sets and perform 10-fold random stratified cross-validation. We also retain the most important features according to the classifier's importance using Recursive Feature Elimination (RFE).

5 DEVICE IDENTIFICATION

We first focus on the adversarial goal G1: recognizing a device name/brand from the metadata of its encrypted Bluetooth traffic. This attack is a stepping-stone for other attacks that aim to infer more fine-grained information such as actions performed by the wearer or applications installed on her device (§6). We recall our assumption that the adversary A_{comm} does not observe the pairing event between a wearable w and the smartphone S, which would give him the device information in plaintext. Instead, we demonstrate how A_{comm} can infer this information from encrypted communication patterns, for instance from devices that are already paired and communicating. Some recent related works already highlight that current BLE devices do not rotate their MAC addresses sufficiently or at appropriate times, enabling tracking [31, 35, 64]. We highlight a deeper problem: the communication patterns (e.g., inter-packet timings) are sufficient to accurately identify and track devices.

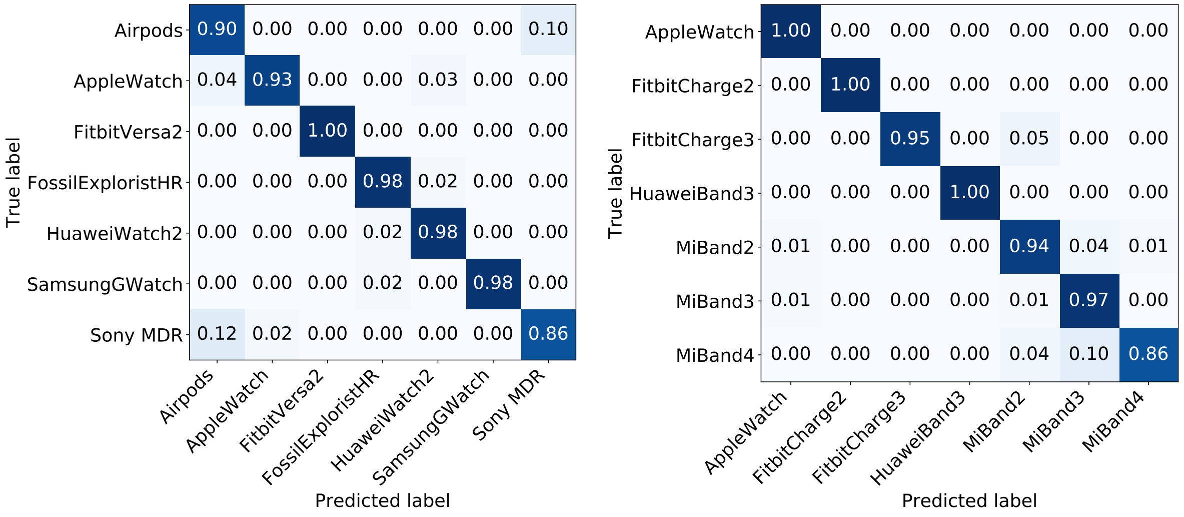
Attack. We use the methodology described in §4 and train two classifiers: one for identifying Bluetooth Classic devices and one for distinguishing BLE devices from their encrypted traffic. We use our captured dataset of encrypted traces with the `device` label as the classifier's target. Initially, we have 10,371 feature vectors across diverse applications/actions of 7 Bluetooth Classic devices, and 329 feature vectors for 7 Bluetooth Low Energy devices. We note that the `AppleWatch` is in both categories as it uses both Bluetooth flavors.

The large sample difference in Bluetooth Classic is due to the automation of two Wear OS smartwatches (§6.2). As the Bluetooth Classic dataset is imbalanced with respect to the number of samples per device, we balance the samples per device label and maintain an equal representation of each device's actions. This gives us between 20 and 60 samples per device, except for the `HuaweiWatch` that has 125 samples (1 per class). In total, the equalized Bluetooth Classic dataset consists of 326 feature vectors. We use a Random Forest classifier with 10 trees (our experiments showed that additional trees were not necessary - Figure 11(a)) that we train using the features described in §4, and we do not limit their depth. We apply Recursive Feature Elimination and keep the most significant 10 features (Figure 11(b)).

Results. For the multi-class classification problems with 7 Classic and 7 LE devices, the classifier's precision/recall/F1 score is 0.96 for Bluetooth Classic and 0.97 for Low Energy (Tables A1 and A2), thus showing that identifying a wearable device from its encrypted traffic is successful. We also find that our classifier accurately distinguishes between different models from the same vendor (i.e., Mi Band 2, 3 or 4, and Fitbit Charge 2 or 3) indicating that each model has distinctive traffic patterns. Figures 4(a) and 4(b) show the confusion matrices. The values in the diagonals are the recall per class.

We perform a feature importance analysis and find that timings are crucial for discriminating among the Bluetooth Classic devices: all three most important features are related to inter-packet timings (Figure 5(a)). This corroborates the findings of Aksu *et al.* [18] who show that the traffic from 6 smartwatches has a distinct inter-packet timing distribution, and is in agreement with fingerprinting results in other domains (e.g., website fingerprinting based on Tor traffic [71]). In the case of Bluetooth Low Energy, the classifier selects primarily size-based features (Figure 5(b)). We postulate that this difference is due to the increased capabilities of Bluetooth Classic devices (i.e., high-end smartwatches) compared to LE devices that consist of simpler devices. Smartwatches support a wide range of possible actions that can generate small or large amounts of data, which makes global-volume-based features less stable than timings that are inherently tied to the OS and the hardware. On the contrary, LE devices support only limited functionalities (e.g., activity tracking or heart rate monitoring), and their communication pattern is inherent to the nature of the application, making size-based features discriminative across devices. Another possible explanation is that due to the absence of a block cipher on the link-layer, BLE packet sizes reveal more information to an eavesdropper. Finally, we observe that the relative feature importance is less pronounced than in the case of Bluetooth classic, thus indicating that the classifier needs more features to successfully distinguish the samples.

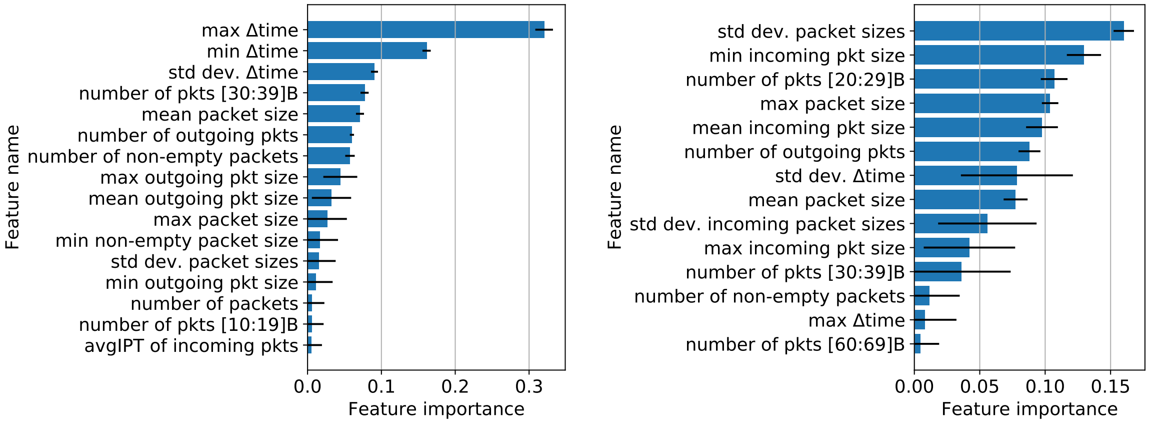
Chipset Fingerprinting. We note that our classifier described above fingerprints a combination of the hardware, the firmware, the OS, and the applications installed. To specifically target the hardware (i.e., chipset manufacturer), we reproduce the same experiment, this time using the Bluetooth chipset manufacturer as the classifier's label. Our goal is to investigate if our classifier learns common patterns from the encrypted traffic recorded by devices that have the same chipset manufacturer, e.g., `Samsung Galaxy Watch` and `Huawei Watch 2`, which both have a `Broadcom` Bluetooth chip or `Mi Band 2, 3` and 4 all equipped with a Dialog Semiconductor chip. Once again, we find that the classifier's performance is high (96% precision/recall/F1 score, Figure 12(a), Table A3), which indicates that there exist stable communication patterns across chipsets. We remark that the limited sample size (i.e., $1-3$ devices per chipset manufacturer) limits the generalization of our conclusions; further analysis is needed in this specific direction. Nonetheless, this result demonstrates the robustness of our methodology, as



(a) Classic devices.

(b) LE devices.

Fig. 4. Normalized confusion matrix per true label for device identification. Values in the diagonal are the recall per class.



(a) Classic devices.

(b) LE devices.

Fig. 5. Feature Importance for device identification. Δtime is the sequence of inter-arrival times. avgIPT_X are the features from Saltaformaggio *et al.* [76] also describing inter-arrival times. The x-axis indicates the relative feature importance.

our earlier results show that our device-identification classifier can distinguish between devices from the same vendor.

Take-Aways. Our experiments confirm that there exist discriminating features across the encrypted traffic of Bluetooth wearable devices; an eavesdropper can use these features to differentiate devices. This raises a question about the level of protection offered by tracking countermeasures, e.g., MAC address randomization, employed by the Bluetooth LE protocol. Moreover, our results show that Bluetooth Classic devices are distinguishable due to their communication time patterns, whereas LE ones have distinct size patterns. We also find that the traffic of devices with the same chipset manufacturer have common patterns. Despite these common patterns, devices from the same vendor can still be distinguished from each other using the device-identification methodology, demonstrating the robustness of our approach. Finally, we remark that device identification is an entry-level attack to other attacks (e.g., action or application identification described next) that aim at inferring more sensitive information about the owners of wearable devices.

6 ACTION & APPLICATION IDENTIFICATION

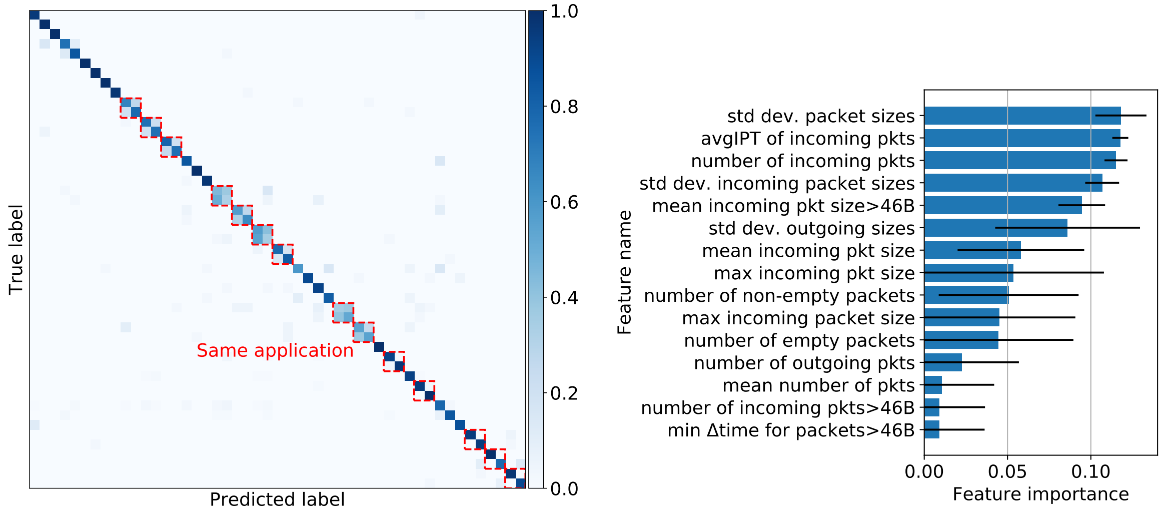
We now consider the adversarial goals G2/G3 and focus on recognizing user-actions from their corresponding encrypted Bluetooth communications. Actions are related to the capabilities of wearable devices: measuring a heartbeat, beginning a workout, tracking a meal or medicine intake, playing music, etc. Our dataset analysis shows that most user-actions — even the mere opening of an application on a smartwatch — result in Bluetooth communication with the paired phone. In this section, we train a classifier to recognize user-actions from the metadata of these encrypted Bluetooth communications.

Due to technical constraints, our experimental evaluation is two-fold and "T-shaped":

1. **Wide-part** (§6.1): We run the attack on all the wearable devices of our testbed (Tables 1 and 2). Since most of them are not automatable due to their proprietary OS/firmware, we use the samples that we manually trigger by performing the appropriate action (e.g., clicking a button on the wearable or performing a short running workout).
2. **Deep-part** (§6.2): We build and use an automation pipeline for Wear OS devices to generate more data. We use this enhanced dataset (1) to identify applications running on a smartwatch, for instance religious or medical applications, (2) to identify fine-grained actions within specific applications, for instance the action "record an insulin injection" in an application that is used to manage diabetes, and (3) to model an attacker capturing traffic over a longer period of time (hours) and attempting to recognize actions and applications in the trace.

Methodology. For our two-fold experimental evaluation, we follow the methodology described in §4 except that we enhance the set of extracted features (see next point). We use the `Action` (or `Application`) label as our classifier's target. In subsections §6.1 and §6.2, we include further details that depend on the specific experimental settings.

Feature Extraction & Classification. We slightly modify the features described in §4. First, we replicate the 3 packet sequences: (1) from Master to Slave, (2) from Slave to Master, and (3) sequences consisting of all non-null, non-ACK packets, and we remove small packets from the copies. We empirically observe that small packets are not useful to the classifier because they are common across applications and actions. After experimenting with different thresholds, we filter out packets smaller than 46B from the 3 new time-series. As before, we extract the min/mean/max/count/standard deviation from these time-series, which leads to 15 additional scalar features. Furthermore, we tweak the features proposed by Liberatore and Levine [60] regarding the counts of packets in certain size ranges. Recall that, for device identification (§5), we used coarse, 10-byte wide buckets. However, our analysis of the Bluetooth traffic concerning user actions shows that some of them produce consistent and unique packet sizes. As a result, we define more fine-grained buckets and record the number of packets with size x, for $x \in [46;\ 1,005]$ bytes. We ignore packets above 1,005 B, the maximum payload size in our dataset. This leads to 960 scalar features replacing the 10 described in §4. Finally, we retain the same timing features as those described in §4. Overall, with our modified approach, we extract 997 features that we feed to our random forest classifier.



(a) Normalized confusion matrix per true label. Red squares regroup actions within one application. Full figure: Figure 13.

(b) Feature importance.

Fig. 6. Application and Action Identification for human-triggered actions ("wide" experiment).

To cope with the increased number of labels (i.e., 49 actions in the "wide' experiment of §6.1 and 56 applications for the "deep" experiment of §6.2), we set the number of trees in the Random Forest algorithm to 30 and we use Recursive Feature Elimination to retain the 50 most important features.

6.1 "Wide" Experiment on All Wearable Devices of our Testbed

We first focus on devices whose Bluetooth traffic capture can not be automated: step counters and fitness trackers (`Fitbit Charge 2-3`, `Huawei Band 3e`, `Mi Band 2-3-4`), and smartwatches (`Fitbit Versa 2`, `Apple Watch`, `Samsung Galaxy Watch`), all with proprietary OS/firmware. We also include manually-triggered actions from Wear OS watches (`Huawei Watch 2`, `Fossil Explorist`), but not machine-automated ones that we use in §6.2. The subset of the dataset that we use in this section consists solely of Bluetooth traces triggered by the corresponding human action. We demonstrate that most actions performed on the wearable devices under examination result in a distinctive encrypted Bluetooth communication, and that an eavesdropper can automatically and passively recognize user actions (e.g., starting a workout) and various events (e.g., the reception of an e-mail/SMS/phone call).

Attack. We do not assume that the adversary knows the device; we train a classifier to infer both the device and the action in one step. A true positive occurs when both the device and the action are guessed correctly. We enumerate and collect 49 actions from the set of wearable devices considered and the companion apps installed on the phone (e.g., `Measure Heartbeat`, `Record Food/Water` intake). The dataset is balanced: for each action, our dataset contains between 20 and 25 samples.

Results. The classifier performance over 49 actions is 82% precision/recall/F1 score (Figure 6(a)). We observe that most actions are recognized with a recall close to one which demonstrates that their corresponding Bluetooth communications have distinct patterns. This includes potentially sensitive user actions such as measuring heart rate, beginning a workout, receiving an e-mail or phone call. Moreover, we note that there exist few classes with lower accuracy compared to others (with precision/recall between 40% and 60%). Our analysis shows that these classes concern actions within the same application, e.g., `EndomondoApp_Running` or `EndomondoApp_Walking`, on the `SamsungGalaxyWatch`. However, this can be a limitation due to the number of samples (20–25 per class); we show in further experiments that fine-grained actions within one application can be inferred with more data (§6.2.3). We also observe that the classifier accurately labels the same action `Running` on devices from the same line of products: `MiBand2`, `MiBand3`, `MiBand4` demonstrating how it performs device and action identification in one step. Finally, we observe that our classifier yields the same accuracy for the devices `Fitbit2` and `Fitbit3` – which only `Sync` with their connected smartphone – thus confirming our Device Identification results (§5): an eavesdropper can recognize the communicating devices based on their metadata.

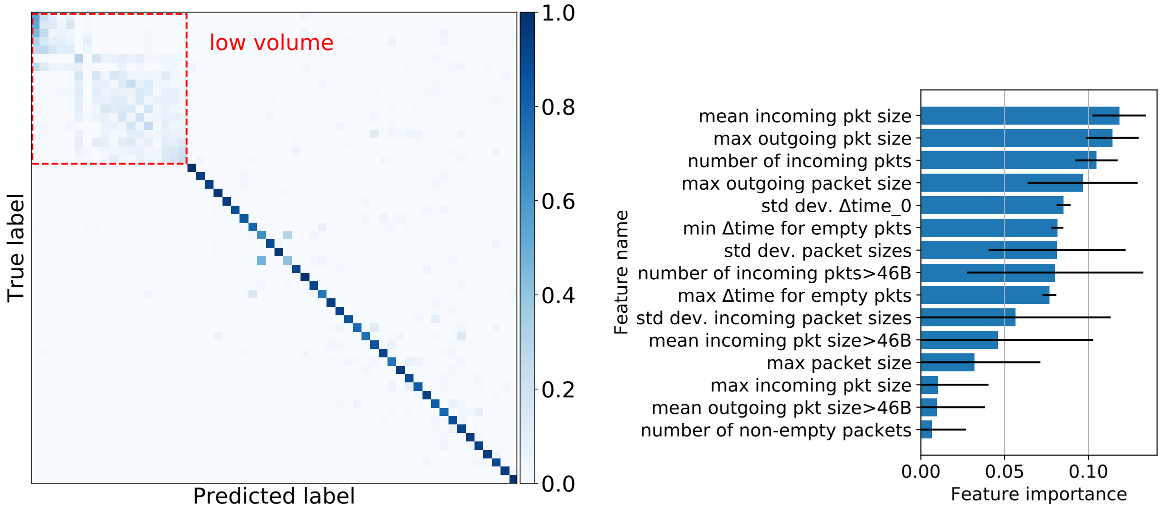
We perform a feature analysis and find that the most important features are based on communication volumes (`number of incoming`, Figure 6(b)) and packet sizes (features 3-8 of Figure 6(b)), with a single highly-ranked feature about timings (`avgIPT of incoming packets`). This is consistent with our initial dataset analysis that demonstrated that various user actions trigger Bluetooth traffic with distinct packet sizes. Finally, we remark that in this experiment, the classifier recognizes the device and the application in one step. In §6.2.3, we improve the classifier's accuracy for identifying actions within the same-application by assuming that device and application identification has already taken place and by tailoring the training of our classifier to one specific application.

6.2 "Deep" Experiment on Wear OS devices

We now perform an in-depth analysis of Bluetooth communications' metadata on Wear OS smartwatches. We use automation to increase the size of our dataset, and we demonstrate the performance of our methodology on various adversarial tasks. To ensure that our synthetic (i.e., computer triggered) actions generate plausible Bluetooth traffic, we restrict ourselves to a one-click action: the opening of an application on the smartwatch. We argue that this should be independent of the wearer's data and inputs and should generalize across users. The first purpose of this section is to demonstrate that the simple opening of a smartwatch application generates (encrypted) Bluetooth traffic patterns that can be accurately recognized by an eavesdropper (§6.2.1). Then, we investigate if the classifier trained by an adversary generalizes and transfers across different smartwatch-smartphone pairs, i.e., to a different setup than that used offline by the adversary (§6.2.2). Furthermore, we build upon our device and application identification attacks, and we target a specific application used to manage diabetes to infer actions within that application. In §6.2.3, we demonstrate how an eavesdropper can passively recognize sensitive, health-related actions (e.g., record an insulin injection) within this specific application. Subsequently, we show how a persistent adversary that continuously monitors the Bluetooth traffic of a user can extract her profile by inferring her application openings and actions over the course of a day (§6.2.4). Finally, we briefly investigate the effect of dataset *aging* on the attacker's classification performance (§6.2.5).

6.2.1 Application Identification. We train our classifier to infer the opening of specific applications on a smartwatch. Identifying app openings has the benefit of being (1) independent of user actions and data more than of the recognition of actions (§6.1), and (2) easier to automate (and hence to train a classifier upon) for an adversary.

We remark that the mere presence of an app can leak sensitive information about the wearer. For instance, medical and well-being applications (e.g., medication reminders, applications to stop smoking) hint that the wearer is concerned with a medical condition, and religious (e.g., prayer time reminders) or news applications with a political orientation can reveal information about the users' beliefs. Even less revealing apps can be useful to a long-term adversary; users naturally install applications based on their interests and behaviors, and the list of apps on their wearable device can be exploited to build personal profiles. We argue that it is difficult to foresee whether the presence of an application is sensitive or not, especially when considering long-term profile building based on data from multiple sources, e.g., for machine-learning-based advertising [47, 69]. We



(a) Normalized confusion matrix for 56 apps, Wear OS. Apps are sorted by increasing median transmitted volume.

(b) Feature importance for application identification, "deep" experiment.

Fig. 7. Normalized confusion matrices per true label for recognizing smartwatch application openings.

envision that as Bluetooth sniffing technologies are becoming less expensive (see our discussion in §8), Bluetooth traffic could become a valuable source of information. We remind the reader that companies are currently experimenting with Bluetooth-based "proximity advertising", a technology used to track users and display local targeted advertisements in transportation systems, airports, and supermarkets [1, 2, 6].

Automation Pipeline & Applications. We use the automation part of our Bluetooth traffic capture pipeline (§4) that consists of a Linux laptop that coordinates with a Windows machine that records traces via a Bluetooth sniffer. Using adb and monkeyrunner [12], the Linux laptop issues synthetic clicks and swipes to a Wear OS smartwatch connected over Wi-Fi. We force the watch to send data over Bluetooth (rather than Wi-Fi) by making sure that the Wi-Fi network does not have Internet access. We do not perform UI fuzzing; we manually specify the clicks and swipes needed to perform the desired actions on the watch. The exact same clicks and swipes are reused to repeat an action. At the time of writing, due to the lack of debugging tools, only Wear OS smartwatches could be automated in the desired way. Devices running Tizen OS should support automation using a variant of adb called sdb [14], however, the current API does not enable it. Our first experiment uses a Huawei Watch 2 (LEO-BX9) running Wear OS 2.16, paired with a Pixel 2 running Android 9.

We select 56 applications from the Google (Wear OS) Play store, favoring top-rated applications per category. Our choice of applications was constrained by the availabilities of apps on the Swiss Play store. Our list includes:

- Religious apps (`DuaKhatqmAlQuran`, `AthkarOfPrayer`, `SalatTime`, `DCLMRadio`)
- Health-related apps (`DiabetesM`, `Medisafe`, `SmokingLog`, `Qardio`, `HeartRate`)
- Lifestyle-related apps (`Lifesum`, `Calm`, `DailyTracking`, `HealthyRecipes`, `SleepTracking`, etc)
- Sport/Activity-related apps (`Endomondo`, `FitWorkout`, `FITIVPlus`, etc)
- Local newspapers (`ChinaDaily`, `WashingtonPost`, `Meduza`, `Krone`)
- Mobile banking and finances (`ABS`, `Mobills`)

- "Local guides"/maps (Citymapper, Foursquare)
- Communication apps (Telegram, Glide, Outlook)

We also include stock applications (e.g., Reminders, Weather, Phone), as well as common applications (e.g., Telegram, Translate) as a control for the sensitive groups and to increase the number of applications. Finally, we include a NoApp label that corresponds to background communications between the smartwatch and the phone.

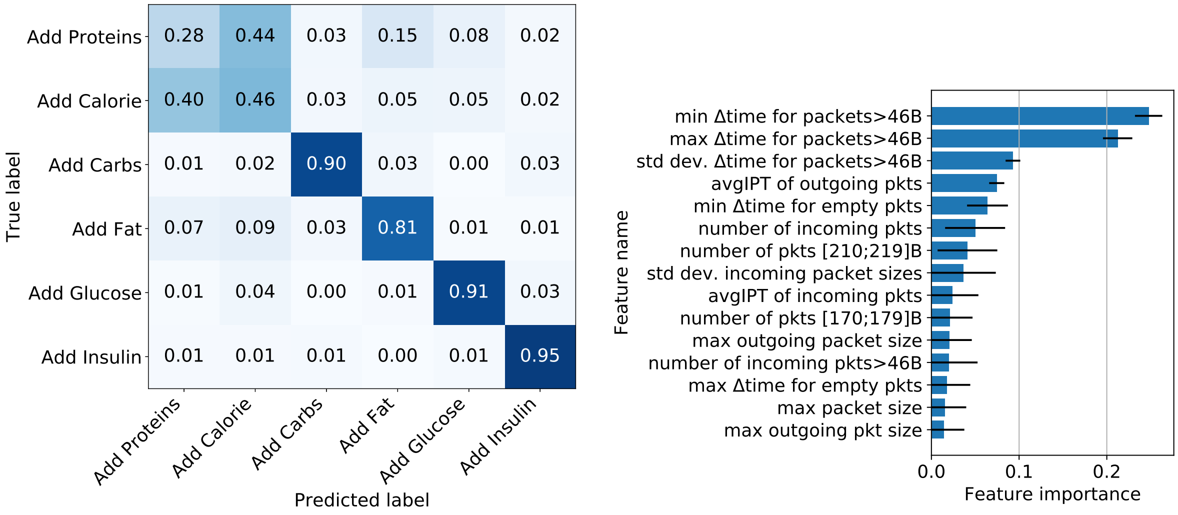
Results. The classifier's performance (precision/recall/F1 score) over the 56 apps is 64%. However, we find that the precision and recall per class varies greatly (Tables A9 and A10). In particular, the majority of apps (38 out of 56 apps) is classified with a mean accuracy close to 1, whereas a smaller subset of the apps is confused among each other by the classifier. Our analysis shows that this concerns apps that trigger minimal Bluetooth communications upon their opening (e.g., Battery, Flashlight); this fact is visible when we order the confusion matrix by the median transmitted volume (Figure 7(a)). We call these apps "low-volume", as they communicate less than 200 bytes for at least 75% of their samples (red dashed square of Figure 7(a)). However, except for Battery and Reminders, which are the unique apps that trigger absolutely no communication, we find that low-volume apps communicate a small amount of information on the Bluetooth layer (Table A7). To further investigate the difference across the two subsets of apps, we train two separate classifiers: one specifically targeting the 18 low-volume apps and another one to distinguish among the remaining 38 non-low-volume apps (Figures 14(b) and 14(c)). The mean accuracy of the first classifier reaches 17% with high variance across the labels, which shows that the data exchanged by these apps is simply too small and/or too variable to be learned by the model. We find that the NoApp label that corresponds to background communications between the smartwatch and the smartphone is part of the low-volume group. This indicates that low-volume apps are not only hard to distinguish among their peers but are also difficult to differentiate from the absence of activity. On the contrary, the non-low-volume apps are recognized by the second classifier with a high accuracy of 90%, demonstrating the practicality of the attack in this case. We observe that — as before — the important features are based on the sizes of packets and the timings, with an unsurprising emphasis on the direction smartwatch→smartphone for the timings (Figure 7(b)).

6.2.2 Model Transferability. Our application-identification attack (§6.2.1) was successful on a single smartwatch-smartphone pair (i.e., Huawei Watch 2 - Pixel 2). However, our attack would require the adversary to train a classifier on the particular pair of devices that the target possesses. In this subsection, we investigate if the trained model generalizes to other devices, upon which the adversary has never used for training.

In more detail, in the previous experiment we use a Huawei Watch 2 (LEO-BX9) running Wear OS 2.16, paired with a Pixel 2 phone running Android 9. In this experiment, we include a new pair of devices: a Fossil Q Explorist HR smartwatch running Wear OS 2.16, paired with a Nexus 5 phone running Android 6. We could not downgrade one Wear OS device to a different version to introduce more variability to our experiment. Also, we could not include the Apple Watch / iPhone pair in the transfer experiment due to a lack of overlap in the apps that we selected and the apps available in the Apple Store. Hence, we leave the question of cross-OS transferability as an interesting future work.

We select 34 apps from the Huawei-Pixel pair that could also be installed on the Fossil-Nexus pair (Table A5). We follow the same methodology, except that we use all the samples collected from one pair of devices as the classifier's training set and the data collected from the other pair as the testing set. We perform our experiments in both directions.

Our results show that the trained model generalizes well: it has a precision/recall/F1 score of 81% when the data collected from the Huawei-Pixel pair is part of the training set and the data collected from the Fossil-Nexus pair is the testing set (Figure 15(a), Table A11). The classifier's precision/recall/F1 score reaches 86% when we perform



(a) Normalized confusion matrix per true label.

(b) Feature importance.

Fig. 8. Fingerprinting fine-grained action within on application: `DiabetesM`.

the experiment in the opposite direction (Figure 15(b), Table A12). Some apps are misclassified: our analysis shows that these are applications that are native to the OS (`Fit Packages`). This is not surprising, given the differences in major Android versions. However, and more importantly, our experiment shows that fingerprinting non-native applications by their (encrypted) Bluetooth traffic is possible independently of the smartphone's OS version, which demonstrates the robustness of our attack methodology.

6.2.3 Fingerprinting Fine-grained Actions Within an Application. We now use the application-opening identification (§6.2.1) as a stepping stone to another attack that aims at inferring potentially sensitive actions within one particular application. For a use-case, we choose the app `DiabetesM` that helps people diagnosed with diabetes to keep track of their meals and medicine intakes. Although the Huawei smartwatch that we use for this experiment is not marketed as a medical device, this information is clearly of medical nature.

We follow the previous methodology and use the automation tool to generate traffic that corresponds to the usage of the `DiabetesM` app. We manually program user interactions (i.e., pressing buttons) within the application and capture the traffic of 6 actions related to the management of meals and medicine intakes: `Add Calorie`, `Add Carbs`, `Add Fat`, `Add Glucose`, `Add Insulin`, and `Add Proteins`. We collect 150 samples per action, map them to feature vectors using the features described in §6 and split them into 80% for training and 20% testing. Then, we train a classifier tailored to this particular app that aims to distinguish among the 6 possible user actions, i.e., we assume that the application's traffic has been already classified as `DiabetesM`.

The overall accuracy of the classifier over the 6 actions is 70% (Figure 8(a), Table A4), which is significantly better than guessing at random. This indicates that actions within a specific application generate distinct Bluetooth traffic that can be fingerprinted by an eavesdropper. More importantly, our results show that the sensitive action `Add Insulin` is recognized with a precision/recall/F1 score of 90/95/92%, respectively, i.e., the encrypted communication patterns generated by this sensitive action "stand out" from other actions of the `DiabetesM` app.

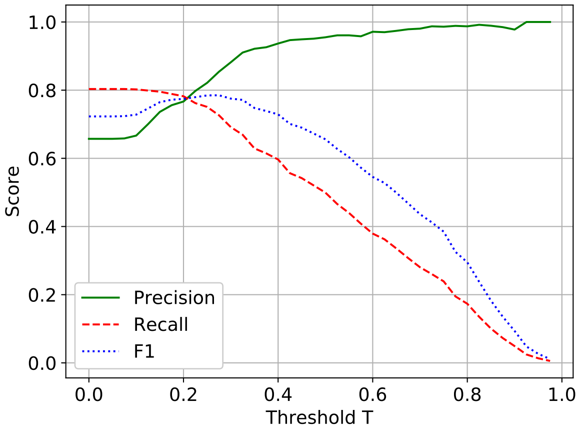


Fig. 9. Attacker's score when classifying events over 24-hour captures. The threshold is the minimum confidence needed to output a prediction.

We here remark that all of these actions are semantically similar: they all update a variable in a database stored on the paired smartphone. We expect that developers could prevent the traffic-analysis attacks by simply padding the traffic corresponding to all of these actions to a constant size. However, the feature analysis reveals that *timings* matter most, not sizes (Figure 8(b)). A closer inspection reveals that a human has to interact with `DiabetesM` in two consecutive steps: (a) by increasing a value (e.g., pressing "record insulin injection"), and, (b) by clicking on a "save" button on a different screen. To our surprise, both actions generate traffic, and the classifier detects the timing differences between the two actions.

First, this highlights a limitation of our experiment. In our methodology, the duration between the press of the first button, the swipes, and then the press of the "save" button are constant and precisely reproduced by the automation framework. A human would produce more variable durations that would be harder to fingerprint by their encrypted traffic. However, we believe that this finding is still an interesting showcase for the capabilities of traffic-analysis attacks: padding each action into a similar message is not sufficient, because the classifier can "count" the number of swipes from the screen containing the save button back to the screen containing the target action. Hence, it is necessary to obfuscate this duration, or rethink the strategy that the application employs to synchronize data with the smartphone.

6.2.4 A Persistent Adversary. We now consider a longer-term adversary that aims at identifying the actions performed on a smartwatch by its wearer over the course of a day. This adversary could be a nosy neighbor or an office eavesdropper, capturing Bluetooth traffic continuously over a long period, and attempting to monitor the habits of the target. This experiment drops a simplifying assumption made in the previous experiments, i.e., that the adversary knew when the traffic related to an action starts and stops. Indeed, all previous experiments used 30-second traffic samples, each corresponding to exactly one action. In this new experiment, the adversary records a continuous traffic capture over 24 hours, and its goal is to output a series of predictions over this period. Furthermore, the adversarial task is slightly tweaked: the prediction can be either a user-action, or the `NoAction` label corresponding to the absence of user activity. Finally, we note that due to their long duration, these traffic captures contain background traffic (updates, synchronization) more than the short 30-sec samples used previously. The goal of the adversary is therefore to distinguish specific app openings from background noise and OS communications.

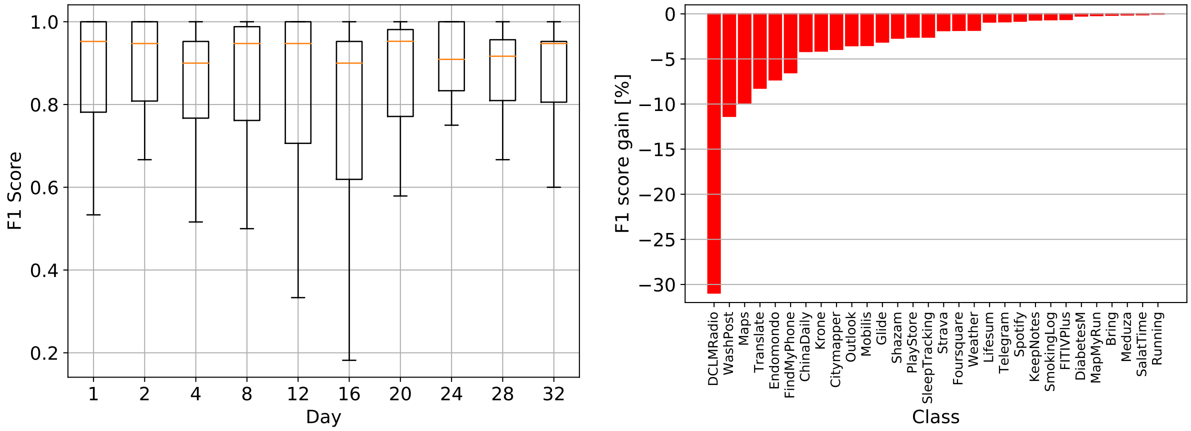
Methodology. We generate application openings and user actions with the HuaweiWatch2 smartwatch, using the automation framework presented in the "deep" experiment (§6.2.1). We simulate one user who wears the watch for 24 hours. Over the course of a day, we model the user's interactions with her smartwatch following a recent user study that quantifies smartwatch usage in the wild [61]. In particular, for each 1-hour slot, the number of interactions is drawn from a probability distribution favoring daytime hours over nighttime ones (Figure 4 of [61], page 389). User actions are not triggered with an equal probability: popular applications such as messaging/e-mails, maps, alarms/clocks, and fitness trackers, are more likely to be triggered than others (Table 6 of [61], page 390). We model this by updating their prior probability to 2× compared to that of non-popular applications. We select 33 high-volume applications from popular categories and enumerate 17 user actions within these applications, e.g., DiabetesM_AddInsulin, HealthyRecipes_SearchRecipe, FitWorkout_Open (Table A6). Individually, each of these actions has a short duration ($\leq$ 20sec). However, these fine-grained actions follow each others in semantic sequences: e.g., we automate the sequence Endomondo_Open, Endomondo_Running, waiting 2min, Endomondo_Close, which is the equivalent of a 2-min workout. The classifier attempts to recognize each individual action (except for _Close actions).

Then, we automate the recording and triggering of actions. Due to technical constraints, we record 20min-captures that we concatenate to form a 24h capture. The parameters of each 20min-capture are drawn from the distribution of the modeled time of the day. Within one capture, the simulation is a simple state machine that loops over the following actions: (1) it flips a biased coin deciding whether to trigger an action or not, and (2) if the outcome is positive, it draws one action at random following the biased probability distribution, runs the action, and waits for a random time defined by the expected number of events in the modeled time of the day. In parallel, the smartwatch and smartphone normally exchange background data.

Attack. To train its classifier, the attacker uses the short 30-sec captures corresponding to the 50 classes (i.e., applications and actions) selected. Moreover, it employs a 51st class that it uses to model noise: this class contains the background communications of the wearable, recorded as NoApp_NoLabel. In addition, we notice that closing an application also generates network communications. The volume exchanged is low, hence they are difficult to classify. However, we observe that when treated as noise, they are useful in helping the classifier distinguish between actions of interest and background traffic. Therefore, we add the classes AppX_Close for all 33 applications of interest. We note that the adversary's dataset is balanced: it contains between 30 and 40 samples per class. Finally, the attacker extracts features, as in the previous experiments (§6.2), and trains a Random Forest classifier.

During testing, the attacker is provided with the uncut 24-hour traffic capture. First, it runs a splitting algorithm that identifies sequences in the capture that possibly correspond to user-actions. This splitting algorithm uses a sliding window and records the times at which the sum of bytes exchanged in the window is greater than 200 bytes, following the criteria for "high-volume" apps presented in §6.2. Then, it classifies the contents of each window; however, it only outputs the most likely class if its probability is greater than a confidence threshold T. If no class meets this criteria, it outputs NoAction.

Evaluation Metrics. We adapt the attack's evaluation metrics to the new task. A true positive corresponds to a correct prediction in the "correct" time-interval, i.e., if the time intervals of the real action and the predicted one overlap. A false positive occurs when the attacker's classifier outputs a label other than NoAction that does not



(a) Training on day 0, testing on day *i*.

(b) Evolution of the F1 score per class, averaged over 32 days.

Fig. 10. Effect of dataset aging on accuracy.

overlap with a real action of the same label. Finally, a false negative is when the classifier misses a real action. Following these definitions, we calculate the classifier's precision/recall/F1 score as usual.

Results. The results are computed over the 72 × 20min = 24 hours of the experiment. We parameterize them with the confidence threshold T, which impacts the overall sensitivity: lower T values result in a higher recall and lower precision, and vice-versa. For the classification task with 51 classes, the classifier's average precision ranges from 0.65 to 1, and its mean recall per class from 0.7 down to 0 (Figure 9). The maximum recall is 83.5%, for a threshold T of 0.1, and the corresponding precision is 74.9%. The maximum precision is 1.0%, for $T = 0.6$, and the corresponding recall is 23.5%. The best F1 score is 0.83 and is achieved at $T = 0.25$. This experiment demonstrates that a persistent adversary successfully recognizes high-volume applications from the absence of activity and accurately classifies them over the course of the day. The different values for the confidence threshold T indicate a precision/recall trade-off. An adversary can choose to optimize its strategy towards one metric or the other (or both) depending on its goals. For instance, an adversary who aims at recognizing, with high precision, an application of interest or a particularly sensitive action (e.g., , `AddInsulin`) could do so at the cost of more false negatives (and lower recall). Whereas, another adversary, e.g., a smart billboard displaying an advertisement to a passer-by, aiming to identify the set of applications and actions of its target (that can help to build a profile) could choose a lower decision threshold and achieve better overall performance.

6.2.5 Dataset Aging. Finally, we briefly explore the effect of dataset *aging*, that is, the loss of accuracy incurred as the adversary uses older datasets. We perform an experiment in which we collect data from applications over 32 days; then, we measure the classifier accuracy when classifying each day's samples with the model trained on the data collected at day 0. We use the 38 high-volume applications presented in Section 6.2, from which we exclude 5 applications that were not successfully automated and did not produce traces over the multiple days of this experiment. We also exclude 3 applications that stopped communicating data after an update over the month. Figure 10(a) shows a boxplot of the F1 score per application over the duration of the experiment. First, we observe that the median F1 score is stable through the month, with small variations between 90% and 95%. The overall performance of the classifier did not vary significantly, even when using traces that are a month old. We do observe that the F1 score of some classes varies through the month (e.g., on days 12 and 16, the lower whiskers indicate that some apps were misclassified more than usual). We investigate the F1 score per class, averaged over the month (Figure 10(b)). We see that the different classes have variable F1 scores, in particular with one application being harder to classify: `DCLMRadio`. This a web radio that loads a large amount of content updated daily. It is unsurprising that training on a single day is not representative of the whole month. Nonetheless, other applications are still classified with high accuracy after 32 days.

Finally, to improve the classification, we also experiment with training the classifier using the data of several days. We use the data collected during the initial three days (instead of just the first), while keeping the same amount of training data (corresponding to a single day). We observe that training on just a few days leads to a performance increase of 3% mean accuracy (from 92% to 95%) over the month. The F1 score of `DCLMRadio`, the worst class, also increases by 14 percentage point (from 68% to 82%; Appendix, Figure 17) . This suggests that an adversary does not need a fresh dataset to perform the attack. Thus, this lowers the cost of the attack by reducing the amount of training needed and by facilitating dataset reuse.

6.3 Summary of the Attacks

Overall, the experimental results of this section demonstrate that different actions (e.g., declaring an insulin shot on a diabetes monitoring application) performed on a wearable device trigger unique Bluetooth traffic. These communication patterns can be recognized by an eavesdropper (e.g., a nosy neighbor or a proximity-based advertiser) to infer the action performed despite the encryption. This holds across the multiple wearable devices such as smartwatches and fitness trackers from diverse vendors. We also find that the mere opening of an app on a smartwatch generates distinguishable traffic, leaking potentially sensitive information that is associated with the presence of the app (e.g., a religious or a political app). For application-openings, we identify a subset of applications that is inherently well-protected against traffic analysis: "low-volume" applications. This hints that minimizing data exchanges is an obvious natural defense against our attacks. Furthermore, our results demonstrate that our attacks generalize well across different devices: we show that a model trained for recognizing application openings on one smartwatch-smartphone pair can be applied with high accuracy on another smartwatch-smartphone pair. This suggests that our methodology can be cost-effective for an adversary. Finally, we demonstrated how a persistent long-term tracking adversary can benefit from our traffic-analysis attacks by employing them to profile users, i.e., infer their habits and actions on their daily lives, and that dataset aging does not significantly affect the attack's performance.

7 PROTECTIONS

In the previous sections (§5, §6), we demonstrated how an eavesdropper capturing Bluetooth communications between a wearable device and its connected smartphone can perform traffic-analysis attacks and infer information such as the device model, the applications installed on it, and the actions performed by the wearer. In this section, we investigate defense strategies against Bluetooth traffic-analysis attacks. We first review the existing types of defenses against traffic-analysis attacks and identify the most popular approaches. Then, we implement these strategies and evaluate them against our traffic-analysis attacks.

High-level Defense Strategies. Before diving into the design of a defense, we remark that *data minimization* is a simple, inexpensive, and valid approach: data that is not exchanged cannot be fingerprinted. Indeed, our "deep" experiment shows that applications with low traffic volumes are naturally better protected against traffic analysis (§6.2). In a similar vein, infrequent "bulky" communications (e.g., syncing a step counter only once a day at midnight) make the adversary's task harder in two ways: by leaking less metadata about timings, and by requiring the adversary to observe the communication at the right time. However, such high-level strategies do not apply to all applications (e.g., interactive applications such as newspapers, radios or music players).

Defense Design. The purpose of this section is not to discover a *perfect* defense: it might not exist or its cost might be unbounded. Rather, we evaluate the effectiveness of common protections against traffic analysis with respect to the attacks that we presented earlier and study the communication overheads that they introduce. Fundamentally, a classifier properly trained can detect even small differences between two network traces. Ensuring that all network traces are perfectly indistinguishable from each other is infeasible. Therefore, we analyze the effectiveness/cost trade-offs introduced by standard defenses and discuss their feasibility for the protection of Bluetooth communications.

Defense Categories. We first perform a brief taxonomy of defenses against traffic analysis. The most active research fields are focused on the Tor network [34, 48, 49, 57, 62, 67, 87, 90, 92] and on IoT traffic/smart-homes [21, 27, 28, 48, 85]. In a different category, recent anonymous messaging protocols often resist traffic analysis against a much stronger global adversary, at the cost of having a high bandwidth [29, 37] or a high latency [22, 43, 59]; indeed, spending time or bandwidth is a fundamental trade-off for achieving traffic-analysis resistant communications [46]. To the best of our knowledge, traffic-analysis defenses have not yet been explored on wearable devices.

We distinguish three defense categories [49]: regularization, obfuscation, and randomization:

- **Regularization** defenses make packet traces harder to distinguish by removing their differences, e.g., by enforcing constant bit-rates and packet sizes [28, 33, 34, 48], by altering the traces into the common closest "super-sequence" of packets [87], or by forbidding duplex communications [90].
- **Obfuscation** approaches aim at confusing the adversary by tweaking the setting, e.g., by hiding traffic into another protocol [62, 92], or by loading two web pages at the same time, as in the case of website fingerprinting [67].
- **Randomization** defenses confuse the adversary by adding randomized dummy traffic [21, 27, 48, 49, 57].

Defenses based on regularization are often easier to reason about and to analyze their formal guarantees, but they have the downside of being more costly than the others. On the contrary, obfuscation approaches consist of more practical defenses that often assume a certain type of adversary, e.g., that cannot de-multiplex encrypted web pages, or recognize Tor traffic hidden as Skype traffic. We do not explore obfuscation strategies as this category does not apply well to wearable devices that only support a few classes of traffic. One possible obfuscation strategy (not explored in this work) could be to split the traffic between Wi-Fi and Bluetooth, for smartwatches that are capable of both. Finally, randomization defenses tend to be the most lightweight. However, their efficacy evaluation is harder and is typically done using the success rate of the state-of-the-art attacks [49, 57].

Defense Evaluation. Our goal is to investigate and understand what degree of protection against Bluetooth traffic-analysis attacks would be provided by a practical and lightweight defense. We remind the reader that our classifiers use features based on timings (Figures 5(a),7(b),8(b)) and packet size distributions (Figures 5(b),6(b)). Thus, we evaluate three orthogonal defenses that mask real sizes and timings, and that inject dummy packets (which achieves both):

(1) `pad`: A lightweight regularization defense. Each Bluetooth packet is individually padded to a maximum size (255B for BLE packets and 1,021B[3] for Bluetooth Classic packets). Per-packet padding hides specific sizes and unlike per-flow padding, it incurs no delay (ignoring the small delay due to transmitted larger packets).

(2) `delay_group`: A regularization defense that delays and groups packets to the next second. This obfuscates fine-grained timing information by imposing a pace. We note that this approach is clearly incompatible with

[3] This corresponds to the max payload of a 3-DH5 ACL packet in Bluetooth Classic.

latency-sensitive Bluetooth communications such as audio streaming, real-time and interactive applications. It does not incur bandwidth overhead.

(3) `add_dummies`: A randomization defense that injects packets at times drawn from a Rayleigh probability distribution. The use of the Rayleigh distribution is inspired by the "Front" part of Front/Glue [49], a state-of-the-art lightweight randomization defense designed for website fingerprinting. We experimentally select 6s for the mean of the Rayleigh distribution, and 300 for the number of dummies we generate (Figure 18). Finally, we sample the size of each dummy from a distribution created with the collected samples. Therefore, this defense assumes that the defender knows a priori the distribution of packet sizes.

To assess the protection level provided by the defenses, we measure the accuracy achieved by the classifier trained by the adversary. We assume that the adversary knows the defense in use and is able to adapt the training of the classifier. To quantify the cost of each defense, we use 5 metrics: the mean delay introduced per packet, the total added duration to the sample, the number of bytes added (both in terms of padding and dummy messages), and the total size overhead in percentage.

7.1 Experimental Results

We apply the various defenses on the Bluetooth traffic traces used for the device identification attack (§5), the "wide" experiment consisting of human-triggered actions on all wearable devices (§6.1), the "deep" experiment consisting of automated apps openings on Huawei Watch 2 (§6.2), and the fine-grained action recognition within the `DiabetesM` application (§6.2.3). This enables us to evaluate the performance of the defenses against multiple adversarial goals and various traffic settings.

Tables 4, 5, 6, 7, and 8 display the performance and cost of each defense against the attacks considered. The accuracy of the attack is averaged over the possible classes, and the defense costs are averaged per traffic sample. Our first immediate observation is that regardless of the attack and the defense, the mean accuracy achieved by the adversary's classifier is still significantly better than random guessing, which indicates that the defenses are far from being "strong" ones that provide cryptographic guarantees. Overall, the cost of each defense lies between 1–23× in terms of data transmission (at most ≈ 400 KB of extra data).

Device Identification. Tables 4 and 5 show that all defenses yield, at best, a moderate drop in this attack's accuracy. However, both flavors of the device identification attack are performed on a small number of devices (7 both for Bluetooth Classic and Low Energy). It is therefore not surprising that hiding the traffic of a device into that of another is a difficult task for the defenses. Among the evaluated defenses, we observe that the `delay_group` is the most effective one: it diminishes the attack's accuracy by 29 percentage points in the case of Bluetooth Classic, and 17 for Bluetooth Low Energy. However, the cost of `delay_group` is prohibitively high (≈ 0.5s delay added per packet) for a defense that is meant to be applied to all the communications performed by a device. Moreover, we find that the defense `pad` is ineffective with both Bluetooth flavors. This is not surprising for the case of Bluetooth Classic where the attacker's classifier relies mostly on timing features (Figure 5(a)). For Low Energy, features that relied on packet sizes (Figure 5(b)) have been replaced by the same top-rated timing features as in Bluetooth classic (`max/min/std of Δtime`, Figure 19(a)). This suggests that these three features are important for device identification, regardless of the Bluetooth flavor and corroborates the findings of Aksu *et al.* [18] on smartwatches. We observe that `add_dummies` is lightweight and reduces the attacker's accuracy by 18 and 12 percentage points for Bluetooth Classic and LE, respectively. However, its performance is not uniform across the classes (Figure 20). The confusion matrix shows that `add_dummies` only moderately protects the 3 `Mi Band 2-3-4` devices and does not protect the others.

"Wide"-Experiment. We evaluate the performance of the defenses against the action identification attack demonstrated in §6.1. Compared to the previous attack, the task is different, and the classifier has to account for more

Table 4. Analysis of the defenses against device identification, Bluetooth Classic devices.

Defense	**Accuracy [%]**	**Delay/pkt [s]**	**Extra dur. [s]**	**Padding [KB]**	**Dummy [KB]**	**Overhead [%]**
No defense	96.3	-	-	-	-	-
pad	93.8	-	-	401.6	-	203.2
delay_group	67.7	0.5	0.2	-	-	-
add_dummies	78.0	-	-	-	92.9	47.0

Table 5. Analysis of the defenses against device identification, Bluetooth LE devices.

Defense	**Accuracy [%]**	**Delay/pkt [s]**	**Extra dur. [s]**	**Padding [KB]**	**Dummy [KB]**	**Overhead [%]**
No defense	97.7	-	-	-	-	-
pad	94.5	-	-	139.0	-	277.1
delay_group	80.6	0.5	0.1	-	-	-
add_dummies	85.8	-	-	-	20.4	40.6

Table 6. Analysis of the defenses against action identification, "wide" experiment.

Defense	**Accuracy [%]**	**Delay/pkt [s]**	**Extra dur. [s]**	**Padding [KB]**	**Dummy [KB]**	**Overhead [%]**
No defense	82.3	-	-	-	-	-
pad	64.1	-	-	272.0	-	270.5
delay_group	52.0	0.5	0.2	-	-	-
add_dummies	64.0	-	-	-	62.5	62.2

Table 7. Analysis of the defenses against application identification, "deep" experiment.

Defense	**Accuracy [%]**	**Delay/pkt [s]**	**Extra dur. [s]**	**Padding [KB]**	**Dummy [KB]**	**Overhead [%]**
No defense	64.4	-	-	-	-	-
pad	27.9	-	-	150.5	-	585.0
delay_group	37.3	0.5	0.4	-	-	-
add_dummies	33.8	-	-	-	46.8	182.0

Table 8. Analysis of the defenses against action-identification in DiabetesM application.

Defense	**Accuracy [%]**	**Delay/pkt [s]**	**Extra dur. [s]**	**Padding [KB]**	**Dummy [KB]**	**Overhead [%]**
No defense	70.4	-	-	-	-	-
pad	61.1	-	-	77.8	-	2374.8
delay_group	60.1	0.5	0.1	-	-	-
add_dummies	61.7	-	-	-	11.8	360.2

labels (i.e., actions). We find that all defenses perform better in general, reducing the attacker's accuracy between 18 and 30 percentage points (Table 6). In particular, the difference in efficiency between pad and delay_group is now of only 12 percentage points, for a cost of 272KB per sample for pad, and 0.5s delay for delay_group. This result suggests that both masking individual sizes or masking fine-grained timing information can help; developers could select the appropriate defense based on the cost (either in bandwidth or latency) that best matches their requirements. Finally, add_dummies performs similarly to pad but with a lower cost (62.5KB per sample versus 272KB for pad). However, add_dummies requires that the distribution of packet sizes is a priori known to generate dummies of plausible size. It is unclear how to compute this distribution for a defense meant to be applied to multiple devices from different vendors. One option would be to combine add_dummies and pad to avoid this requirement (we experimented with it and observed better effectiveness at a higher cost). Finally, we note that the protection provided by the defenses is not uniform (we provide an example with add-dummies in Figure 21(a), other defenses yield similar results), but unlike the "deep" experiment (§6.2), the precision/recall per class does not seem correlated with the transmitted size. Similarly, the cost of padding varies greatly with the classes: it has a mean of 272KB added, but a median of only 96KB and a standard deviation of 650KB. The costs soar up to 4.3MB with streaming applications such as AppleWatch_PhotoApp_LiveStream or PhoneCallMissed. We note that the defenses delay_group and add_dummies are more consistent, with a standard deviation of, respectively, 0.04s of delay per packet and 7KB of dummy traffic.

"Deep"-Experiment. The traffic in this experiment consists of automated app openings on a Wear OS smartwatch. For this type of traffic, we observe that all three defenses perform similarly, reducing the attacker's accuracy by 31–36 percentage points down to ≈ 30% mean accuracy (Table 7). This highlights that on a specific class of traffic, i.e., with more homogeneous traffic, the defenses are more efficient. In this case, add_dummies is the least expensive defense, requiring 46.8KB of dummy traffic/sample, which is in the range of what the heaviest applications naturally use (Table A8). A deeper analysis of the results also shows that all defenses successfully confuse the attacker for "medium-volume" apps (Figure 21(b)). "High-volume" applications still stand out, but the gradual hiding visible on Figure 21(b) suggests that increasing the parameters of the defense, e.g., injecting more dummies in add_dummies, could potentially protect better such applications. However, this protection would come at an even higher cost for the applications that transmit smaller amount of traffic. As in the "wide" experiment, we observe that pad has a highly variable cost ranging from 6KB to 1.5MB. We observe the latter on the opening of the Camera, which suggests that pad is not adapted for constant-traffic. As before, delay_group and add_dummies have a consistent cost across labels.

Fine-Grained Action Fingerprinting on DiabetesM. In this case, we observe that all 3 defenses pad, delay_group, add_dummies are only moderately effective, reducing the attacker's accuracy by ≈ 10 percentage points. One possible explanation is the increased number of samples (150/label) compared to the previous experiments (25/label) that enable the adversarial classifier to adapt better to the defenses. In §6.2.3, we highlighted that timings were of importance to classify fine-grained actions in the application DiabetesM. However, in this case the attacker fingerprints a combination of the sizes and the timings, as the feature importance on delay_group defended traces reveals (Figure 19(b)). In this experiment, we observe that all three defenses have a consistent cost across labels.

7.2 Summary of the Defenses

Our experimental evaluation of defense approaches, such as regularization and randomization, against the traffic-analysis attacks presented in this work yields some interesting insights. First, we find that these defenses achieve only a limited protection against our traffic-analysis attacks: although they do reduce the attacks' accuracy, the classifier trained by an eavesdropping adversary still performs significantly better than a random guess. This indicates that even the defended Bluetooth traffic traces contain useful information for an adversary. At the same time, the costs introduced by the defenses are high: to achieve their small levels of protection they introduce additional traffic and/or delays reaching an overhead in the range of 1× to 23× and delays up to 1s per packet. This raises a question about the applicability of such defenses for Bluetooth applications running on current wearable devices. Furthermore, we find that the various defenses behave differently, depending on the adversarial task. In particular, our results show that defending against application or action identification is somewhat easier than device identification, thus indicating that global traffic patterns are the hardest to hide. Additionally, our evaluation shows that the defenses are not *fair*: we find that they do not provide the same level of protection across applications or actions (for instance, apps that communicate a lot are not well protected) and their costs are variable across applications or actions (we observe that applications that stream information have high padding costs, e.g., Camera and fitness applications for workout monitoring). Finally, our empirical evaluation of these defenses confirms the robustness of our attacking methodology as depending on the adversarial task and the defense, our classifier adapts its important features. For instance, Bluetooth Low Energy device identification relied mostly on packet sizes (Figure 5(b)), unlike Bluetooth Classic that used mostly timings (Figure 5(a)). However, when we apply per-packet padding to the Low Energy traces, the classifier adjusts and gives higher importance to timings (Figure 19(a)). Overall, our experimental results highlight the need for the design and evaluation of novel approaches for defending against traffic-analysis attacks on Bluetooth communications.

8 DISCUSSION

Ethical Considerations. We provided every device manufacturer and app developer mentioned in this paper with our findings prior to the publication of this document. To minimize the risk of misuse, we will make the dataset available only for research purposes upon request [30]. We discarded the traffic from other devices in the dataset.

Impact of the Attacks. The traffic-analysis attacks presented in this work can be used to infer information from Bluetooth communications, despite the use of encryption. Device identification enables tracking users only by observing encrypted communications, thus defeating MAC address randomization. This does not require observing any pairings/paging events or plaintext identifiers. Device identification can also facilitate active attacks by revealing the model and version of a communicating device. We note that advertisers already use Bluetooth and Wi-Fi signals to passively and actively locate users (e.g., consumers in a store) [4, 7]. Similarly, application and action fingerprinting leak sensitive actions performed by the wearer, e.g., the recording of an insulin injection or a heartbeat measurement. On a side note, Apple Watch has an "arrhythmia alert" feature that continuously measures the heart beat on the watch and sends notifications to the phone in case of irregular patterns that could indicate a stroke. We could not simulate arrhythmia events, but all the evidence we have from our experiments suggests that such an action could be fingerprintable without an appropriate defense mechanism. Therefore, a passive observer could identify users' susceptibility to heart attacks over the Bluetooth network, despite the encryption. Finally, application-opening identification and action identification can be exploited to build user profiles and to serve targeted advertisements, as it is already the case with Bluetooth-based "proximity advertising" [1, 2, 6].

Cost of the Attack. The overall cost of the attack consists of purchasing a set of devices of interest (including both wearable devices and smartphones). These devices are consumer-grade hardware that have accessible prices. We also showed in §6.2.2 that the adversary does not have to train on every combination of devices and applications. We suspect that after collecting data from enough devices, actions on new hardware can become classified without the overhead of training. A counter-argument is the *aging* of the dataset, which could force the adversary to re-train often. We briefly demonstrate in Section 6.2.5 how a dataset can be used over at least a month, but further study is needed to understand how quickly the usefulness of a dataset degrades. We note that in other domains such as website fingerprinting, attacks have been successful with datasets that were several years old [79]. We expect wearable devices' firmware, OSes and applications to change at a slower rate than websites.

Bluetooth Sniffing Technologies. The adversary also needs a reliable Bluetooth sniffer. The most accurate models are so-called "wide-band" scanners (e.g., the Ellisys Vanguard [9] or the Frontline Sodera [15]). These models ignore the Bluetooth frequency hopping and concurrently capture the traffic of all channels. The complexity and broad functionality of these devices comes at a high price ($\approx$ 50K USD). However, recent research has demonstrated that similar results can be achieved using less expensive Software-Defined Radios (SDRs) [40, 41, 82]. For instance, Cominelli *et al.* built an SDR sniffer that works on a single Ettus N310 board ($\approx$ 10K USD) or two Ettus B210 boards ($2 \times 2,000$ USD) [40]. Finally, there also exists a consumer-grade class of cheaper, less accurate Bluetooth sniffers (e.g., Ubertooth, $\approx$ 100 USD). They only listen on one channel at a time. These low-end scanners attempt to *follow* an active connection by brute-forcing the hopping pattern parameters [75]. When successful, this enables an inexpensive scanner to accurately capture all traffic simply by "hopping along" with the pair of communicating devices. In practice today, this process is still imprecise and many packets are missed. Nonetheless, researchers have shown that using two synchronized Ubertooth scanners leads to improved Bluetooth traffic capture rate [19, 20]. Although this work uses a commercial Bluetooth sniffer, there is an ongoing research trend focusing on less expensive, accurate Bluetooth sniffing.

Impact of Packet Loss on the Attacks. In our experiments, we use a high-end sniffer that is co-located with the target devices. In this configuration, the sniffer has close to 100% packet capture rate. In practice, lower-end sniffers will suffer packet loss. Collisions from other devices and the distance between the target and the eavesdropper also increase loss. We briefly investigate how the attack accuracy varies with degraded capture conditions.

First, we decouple the attacker accuracy, the packet loss and the capture conditions, and we explore the attack accuracy versus the loss rate only. This loss can stem from many real-life parameters (distance, noise, multipath interference, the quality of the eavesdropping device, etc.). As a generic approach, we study the effect of uniform packet loss on our dataset. We simulate packet loss on the captured traces and re-run the application identification ("deep") experiment (§6.2.1). We apply a uniform packet loss by dropping individual packets with a given probability. We then use the methodology already presented (we split the traces into train/test and compute the average classification accuracy). The experiment is repeated 10 times per loss rate. We observe (Appendix, Figure 16) that even with 50% packet loss, the loss in accuracy is only 10 percentage points, with the mean accuracy dropping from 64% to 54%. For high-volume apps, the mean accuracy drops from 90% to 77%. This experiment indicates that the approach is robust to packet losses: even when missing every other packet, the attacker is able to classify with significant accuracy.

Due to the difficulty of relating the various capture conditions to the loss rate, we only discuss experimental results generated by Albazrqaoe *et al.* with the most common inexpensive Bluetooth sniffers: The authors describe how a single Ubertooth, when placed 10m away from the target device and in the presence of significant 802.11 interference [19], has between 25% and 50% packet loss. In similar conditions, a BlueEar sniffer (composed of two synchronized Ubertooth) maintains packet losses below 10% [19]. This observation suggests that the attack could also be performed with inexpensive Bluetooth sniffers.

Other Attacks. There exist a number of *active* attacks that can break the confidentiality of Bluetooth communications [24, 26, 86, 93, 95]. There are also attacks against the MAC randomization mechanism employed by the Bluetooth protocol [31, 31, 35, 64, 96]. Currently, these attacks are more economical to run than our traffic-analysis attacks that require a significant effort for training the adversarial machine-learning classifiers. However, these attacks have already received significant visibility, and we expect that the Bluetooth Special Interest Group and device manufacturers will soon take them into account and apply the necessary countermeasures. We remark that our approach is complementary to these attacks and will be applicable even when the above attacks are patched. Finally, we remind the reader that unlike these works, ours considers a weaker adversary who passively observes ongoing communications.61

Implementation of Defenses. A defense against traffic analysis could be implemented at different layers of the stack involved with Bluetooth communications: the Bluetooth protocol, the OS, or the applications. An implementation in a lower layer, e.g., the Bluetooth stack, would provide application transparency, but specifying a single defense strategy that works across devices and applications without a prohibitive cost seems challenging. On the contrary, application developers could protect against in-application actions fingerprinting by enumerating the data exchanges and ensuring that the traffic from sensitive actions "blends-in". In this case, all sensitive actions would need to have the same patterns as some other non-sensitive actions (or ideally, all other actions). Nonetheless, such an approach would only provide local (i.e., in-application) protection. To make the fingerprinting of applications and cross-application actions more difficult, the various developers would need to coordinate with each other. Otherwise, a defense can easily become itself a fingerprint if only one or a few applications implement it. Therefore, another interesting possibility is to create "anti traffic-analysis policies" in the OS of wearable devices. Apps could request a particular defense strategy that matches their requirements in terms of latency, bandwidth, and battery usage. Meanwhile, the operating system could standardize and maintain defense strategies transparently making their deployment easy for developers.

9 LIMITATIONS

Moderate Testbed Size. Our testbed consists of only 13 devices. Although this number is modest, our experiments incurred significant human costs in operating these non-automatable devices. Our primary dataset consists of 98 hours of raw recording. Each of the 2,215 30-second samples was recorded by a human and required sometimes minutes of resetting the devices, not to mention performing the physical activities corresponding to the action captured. We made a best-effort to cover a comprehensive and diverse set of wearable devices from popular vendors. We are hopeful that future work will further generalize the attack to more devices, which we could only hint towards with our transferability experiment (§6.2.2). We open-source the automation framework built to collect traces from Wear OS devices to facilitate further research on the topic [30].

Closed-world Scenario. Our setting corresponds to a "closed-world" scenario where the adversary can model all possible actions/applications of the devices available in our testbed. This could be justified due to the small number of applications currently available on wearable devices. In future work, we plan to explore the performance of the attacks "in the wild", e.g., by collecting real Bluetooth traffic traces around a campus or a gym and attempting to classify them.

Method Scalability. We demontrated an example in which a model trained on a pair of devices can be used for classifying actions of other devices (§6.2.2). However, this experiment has been limited to two pairs of devices, and it is unclear whether a unique model could be successfully trained to classify actions originating from many classes of devices. While we expect it to be the case for similar devices, our preliminary results show that a model trained to recognize applications on an Apple watch does not perform well when used to recognize applications on an Android device, highlighting that an adversary should take into account heterogeneous devices when training her classifier.

Environment of the Capture. In this paper, we did not experiment with the range of capture and kept the sniffer close-by to the devices (max $\approx$ 2m). Bluetooth Classic and Low Energy have a maximum theoretical range that greatly varies depending on the Bluetooth flavor, the encoding, the sender/receiver's antenna gains and the transmitted power [8]. For consumer devices, the range under optimal conditions is between 50 and 100m and, we estimate, from meters to tens of meters under realistic conditions. Furthermore, our attacks were conducted in a single environment that is fairly noisy (with tens of active Wi-Fi and Bluetooth devices in vicinity). We suspect that a less noisy environment (e.g., a home, in the case of a nosy neighbor) would produce cleaner traces that are easier to train upon, facilitating the attacks, but further study is needed to understand the impact of noise and collisions on traffic analysis.

10 RELATED WORK

Bluetooth Eavesdropping. The ability to sniff Bluetooth communications is essential for performing traffic-analysis attacks. The first open-source Bluetooth Sniffer was BlueSniff [80]: This work demonstrates how to retrieve the MAC address of communicating Bluetooth Classic devices and how to recover the hopping sequence. Similar results are later shown on Bluetooth Low Energy [75]: using an Ubertooth device, Mike Ryan showed how to recover the hopping sequence and eavesdrop on a single BLE connection. The author also demonstrates that a pairing done with JustWorks or a 6-digit PIN can be decrypted. Subsequently, Albazrqaoe *et al.* used two synchronized Ubertooth devices to obtain a capture accuracy greater than a single Ubertooth [19, 20]. Finally, Cominelli *et al.* rely on software-defined radio (SDR) to concurrently capture Bluetooth Classic traffic on all channels [40], at a lower cost than full-band commercial sniffers. Their most recent work uses a GPU to process BLE traffic in real-time [41].

Bluetooth Traffic Analysis. There exist few related works that perform traffic-analysis attacks on Bluetooth communications. This is possibly due to the non-existence of reliable, inexpensive Bluetooth sniffing tools in the past. Closest to ours is the work by Das *et al.* [45]. They focus on 6 Bluetooth Low Energy fitness trackers in a gym. First, they demonstrate that BLE traffic is correlated with the wearer's movements, thus making it possible to infer if the wearer is idle, walking, or running. Second, they show how the traffic is linked to the gait of the wearer, and that the encrypted traffic is enough to recognize a person with 97.6% accuracy across 10 users. Acar *et al.* infer user actions in a smart home using a layered traffic-analysis attack [17]. Their methodology is similar to ours: they first perform device identification and then use it as a stepping stone to further infer device states and user activities. However, their IoT testbed consists of only one device that communicates using Bluetooth (a smart BLE light bulb). To the best of our knowledge, there is no work performing traffic-analysis attacks on Bluetooth Classic communications.

Bluetooth Device Fingerprinting/Tracking. Several works focus on Bluetooth device fingerprinting, i.e., device identification and tracking. Their goal is either to propose an authentication mechanism, e.g., to identify MAC spoofing, or to demonstrate an attack, e.g., BLE device tracking despite the MAC address rotation. Our device-identification attack (§5) falls into the second category; however, we only use it as a first step towards the rest of our contributions (application and user-action identification) that are orthogonal to this category of related works.

On the defense side, Aksu *et al.* create a testbed composed of 6 Bluetooth Classic smartwatches connected to a single smartphone, and they demonstrate that the smartwatches can be identified via their communications' timings [18]. However, their model is different and they do not sniff Bluetooth traces in the air, rather collect them using the Bluetooth Host Controller Interface (HCI) log on the smartphone. Huang *et al.* propose BlueID [56], a system that prevents identity spoofing by fingerprinting the clock of the master device.

Concerning the attacks, Zuo *et al.* demonstrate that Bluetooth Low Energy devices can be recognized by plaintext identifiers found in their communications [96]. They suggest application- or protocol-level solutions to better rotate static identifiers. We note that their solutions would not thwart our device identification attack that works on encrypted traffic (§5). Becker *et al.* use static identifiers differently [31]: they demonstrate that the rotation of MAC address and other static identifiers are not synchronized, which enables defeating the MAC randomization. More generally, Celosia and Cunche examine BLE devices in the wild and show that they fail to properly rotate their MAC addresses, thus enabling tracking [35]. Then, Korolova and Sharma show that the "nearby devices" list that Bluetooth devices maintain and which most applications can obtain, can be used to track users across applications [58]. Targeting more specifically Apple's Continuity protocol, Martin *et al.* reverse-engineer the protocol and find flaws that defeat MAC address randomization and that leak information about the device types and user activities [64]. Similarly, Celosia and Cunche also reverse-engineer the protocol and demonstrate how it reveals information about human activities in a smart home [36].

Other Bluetooth Attacks and Tools. There exist other attacks on the Bluetooth protocol that are orthogonal to our work; for instance, active attacks, or protocol-specific attacks on wearable devices. We note that fixing these will likely not affect our higher-level attacks that rely on Bluetooth communication metadata.

We first list attacks on the Bluetooth protocol and implementations. Antonioli *et al.* demonstrate how to break the key negotiation protocol of Bluetooth Classic [26], forcing it to a 1-byte entropy encryption key. The same authors reverse-engineer and identify vulnerabilities in Google Nearby Connections [23], a protocol that uses a combination of Bluetooth Classic, Low Energy, and Wi-Fi for short-distance transfers. Then, they also exploit role-switching (slave/master) and legacy pairings to perform a Man-in-the-Middle on Bluetooth

Classic [24]. Wu *et al.* present an active spoofing attack on Bluetooth Low Energy by exploiting the reconnection procedure [93], whereas Wang *et al.* demonstrate an active attack to bypass Bluetooth Low Energy authentication and encryption [86]. Finally, Zhang *et al.* show a downgrade attack based on Bluetooth Low Energy's Secure Connection Only (SCO) mode [95].

In the category of Bluetooth tools, Mantz *et al.* present InternalBlue [63], a Bluetooth experimentation framework that enables patching the Bluetooth firmware of Broadcom chips. Similarly, Classen and Hollick show how to analyze Bluetooth communications using consumer devices [38], while Ruge *et al.* design an emulation framework and perform fuzzing to uncover vulnerabilities [74]. In particular, the authors find unattended Remote Code Executions (RCE) on some Bluetooth chips.

Focusing on wearable devices, Classen *et al.* perform an in-depth security analysis of the Fitbit ecosystem [39], analyzing the firmware and the application of a fitness tracker. They find vulnerabilities that enable flashing malware, disabling encryption, and extracting private information about the users. Hilts *et al.* perform a comparative analysis of the security and privacy of fitness trackers [55]; they highlight security vulnerabilities and issues with their data policies.

Other Traffic-Analysis Attacks. Danezis and Clayton first introduced traffic analysis in modern digital communications [44]. Since then, Tor traffic has been the primary target of such attacks due to its popularity and its strong threat model [34, 53, 67, 78, 79, 88–90]. Among the most well-known attacks, are the "k-Nearest Neighbors" by Wang *et al.* [87], CUMUL by Panchenko *et al.*, which relies on cumulative sums of bytes to create fingerprints [66], and k-Fingerprinting, by Hayes and Danezis, which uses a combination of random forests and nearest neighbors for classification [53]. The most recent attacks rely on deep-learning approaches [78] and N-shot learning [79] to achieve higher accuracy.

Traffic-analysis attacks have also been applied on TLS traffic, for instance, to recognize a streamed video [72, 73, 77] or to identify the operating system and the applications [65]. Many recent related works focus on IoT traffic and smart homes [17, 21, 27, 28, 81, 85]; while these works often employ a similar attack methodology, they do not consider the same ecosystem of devices, applications and actions. We focus on devices and applications that have access to live fitness- or health-related information of the wearer, both in a fine-grained manner and over a long period of time. Our contribution is the first work to perform an in-depth analysis of the fingerprintability of devices, applications and user actions in this ecosystem. The different nature of the traffic also has a direct impact on the design of defenses. Similarly, traffic-analysis attacks have been studied in the context of smartphones to identify applications [83, 84, 94] or user activities (e.g., sending an e-mail or browsing a web page) [42, 76]. However, these attacks are not all equal: The majority of them use network- or transport-layer headers [42, 76] or application layer headers [27, 28, 83, 84] and can use IP-based flow separation to facilitate the attack. Therefore, they assume an adversary that is already on the target network. On the contrary, some works consider a weaker adversary and perform the attack using 802.11 Wi-Fi frames [81, 94] or the metadata of tunneled traffic [21], which cannot be easily de-multiplexed per device, application or action. Due to the Bluetooth operation, our attacks fall in the latter category.

Defenses Against Traffic-Analysis Attacks. A straightforward but costly defense against traffic analysis is to enforce identical communication patterns across the classes that the adversary aims to identify. This task becomes increasingly complex and costly with a growing number of classes, hence this technique applies only on a small scale, for example in one particular application. Same as for the attacks, most defenses target Tor traffic [34, 49, 57, 62, 67, 87, 90, 92] or the traffic from IoT devices and smart homes [21, 27, 28, 48]. We distinguish three defense categories: Regularization defenses make packet traces harder to distinguish by removing their differences, e.g., by enforcing constant bit-rates and packet sizes [28, 33, 34, 48], by altering the traces into the common closest "super-sequence" of packets [87], or by forbidding duplex communications [90]. Obfuscation approaches aim at confusing the adversary by tweaking the setting, e.g., by hiding traffic into another protocol [62, 92], or by loading two web pages at the same time in the case of website fingerprinting [67]. Randomization defenses confuse the adversary by adding randomized dummy traffic [21, 27, 48, 49, 57]. To the best of our knowledge, our work is the first to investigate the performance of regularization and randomization defenses against Bluetooth traffic-analysis attacks.

Traffic analysis is also a concern for anonymous communications systems that aim at enforcing similar traffic patterns across *participants*. The adversary considered in this case is a stronger global adversary and anonymous communication systems provide protection at the cost of a high bandwidth [29, 37] or a high latency [22, 43, 59]. It has been shown that spending either latency or bandwidth is a fundamental trade-off for traffic-analysis resistance [46]. In the setting of the attacks presented in this paper, this trade-off does not apply; our goal is not to hide the source of the communication.

11 CONCLUSION

In this work, we have shown that encrypted Bluetooth communications between a wearable device and its connected smartphone leak information about their contents: a passive adversary can infer sensitive information by exploiting their metadata via traffic-analysis attacks. Our empirical evaluation on a Bluetooth (Classic and Low Energy) traffic dataset generated by a diverse set of wearable devices demonstrates that an eavesdropper can accurately identify communicating devices to their model number, recognize user activities (e.g., health monitoring or exercising), the opening of specific applications on smartwatches and fine-grained user actions (e.g., recording an insulin injection), and extract the profile and habits of the wearer. Our experimental analysis of common defense strategies against our traffic-analysis attacks, such as padding or delaying packets, and injecting dummy traffic, show that these do not provide sufficient protection, and that they introduce significant costs for Bluetooth communications. Overall, this research highlights an open problem regarding the confidentiality of Bluetooth communications and the need for designing novel efficient defenses to address it.

ACKNOWLEDGMENTS

This work has been made possible by the help of Friederike Groschupp and Stéphanie Lebrun. We also wish to thank Daniele Antonioli, Sylvain Chatel, Jiska Classen, Ricard Delgado, and David Lazar for the constructive discussions and feedbacks on the drafts. We are grateful to the "Centre Suisse d'Electronique et Microtechnique" (CSEM) for providing us with the Ellisys Bluetooth sniffer. This work was supported in part by grant 200021_178978/1 (PrivateLife) of the Swiss National Science Foundation (SNF). Some illustrations in the figures have been made by the artists freepik, eucalyp and smashicons (flaticon.com).

REFERENCES

[1] 2015. London's black cabs are to be fitted with Bluetooth enabled beacon technology. https://www.thedrum.com/news/2015/09/07/london-s-black-cabs-are-be-fitted-bluetooth-enabled-beacon-technology-. Accessed: 2020-09-20.
[2] 2016. Proxbook: Proximity Marketing in Airports and Transportation. https://unacast.s3.amazonaws.com/Proxbook_Report_Q3_2016.pdf. Accessed: 2020-09-20.
[3] 2018. Content of Premarket Submissions for Management of Cybersecurity in Medical Devices. https://www.fda.gov/media/119933/download. Accessed: 2020-06-29.
[4] 2018. Google can still use Bluetooth to track your Android phone when Bluetooth is turned off. https://qz.com/1169760/phone-data/. Accessed: 2020-03-05.
[5] 2019. Carriers selling your location to bounty hunters: it was worse than we thought. https://www.theverge.com/2019/2/6/18214667/att-t-mobile-sprint-location-tracking-data-bounty-hunters. Accessed: 2020-03-05.
[6] 2019. In Stores, Secret Surveillance Tracks Your Every Move. https://www.nytimes.com/interactive/2019/06/14/opinion/bluetooth-wireless-tracking-privacy.html. Accessed: 2020-09-20.
[7] 2019. TfL introduces wifi tracking to improve ads. https://www.thedrum.com/news/2019/05/22/tfl-introduces-wifi-tracking-improve-ads. Accessed: 2020-03-05.

[8] 2020. Bluetooth Range Calculator. https://www.bluetooth.com/learn-about-bluetooth/bluetooth-technology/range/#estimator. Accessed: 2020-11-09.

[9] 2020. Ellisys Bluetooth Vanguard Advanced All-in-One Bluetooth Analysis System. https://www.ellisys.com/products/bv1/. Accessed: 2020-09-20.

[10] 2020. Forecast unit shipments of wearable devices worldwide from 2017 to 2019 and in 2022 (in million units), by category. https://www.statista.com/statistics/385658/electronic-wearable-fitness-devices-worldwide-shipments/. Accessed: 2020-10-28.

[11] 2020. LiveRamp Customer Onboarding. https://liveramp.com/our-platform/data-onboarding/. Accessed: 2020-03-05.

[12] 2020. MonkeyRunner. https://developer.android.com/studio/test/monkeyrunner/. Accessed: 2020-03-13.

[13] 2020. The rise of employee health tracking. https://www.bbc.com/worklife/article/20201110-the-rise-of-employee-health-tracking. Accessed: 2020-11-13.

[14] 2020. sdb. https://developer.tizen.org/development/tizen-studio/web-tools/running-and-testing-your-app/sdb. Accessed: 2020-11-104.

[15] 2020. Sodera Series of Bluetooth Analyzers. https://fte.com/products/sodera.aspx. Accessed: 2020-10-19.

[16] 2020. Ubertooth. https://github.com/greatscottgadgets/ubertooth. Accessed: 2020-11-09.

[17] Abbas Acar, Hossein Fereidooni, Tigist Abera, Amit Kumar Sikder, Markus Miettinen, Hidayet Aksu, Mauro Conti, Ahmad-Reza Sadeghi, and Selcuk Uluagac. 2020. Peek-a-Boo: I See Your Smart Home Activities, Even Encrypted! *Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)* (2020). https://doi.org/10.1145/3395351.3399421

[18] Hidayet Aksu, A Selcuk Uluagac, and Elizabeth Bentley. 2018. Identification of wearable devices with Bluetooth. *IEEE Transactions on Sustainable Computing* (2018).

[19] Wahhab Albazrqaoe, Jun Huang, and Guoliang Xing. 2016. Practical bluetooth traffic sniffing: Systems and privacy implications. *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys)* (2016).

[20] Wahhab Albazrqaoe, Jun Huang, and Guoliang Xing. 2018. A Practical Bluetooth Traffic Sniffing System: Design, Implementation, and Countermeasure. *IEEE/ACM Transactions on Networking* (2018).

[21] Ahmed Alshehri, Jacob Granley, and Chuan Yue. 2020. Attacking and Protecting Tunneled Traffic of Smart Home Devices. *Proceedings of the Tenth ACM Conference on Data and Application Security and Privacy (CODASPY)* (2020).

[22] Sebastian Angel and Srinath Setty. 2016. Unobservable communication over fully untrusted infrastructure. *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)* (2016).

[23] Daniele Antonioli, Nils Ole Tippenhauer, and Kasper Rasmussen. 2019. Nearby Threats: Reversing, Analyzing, and Attacking Google's' Nearby Connections' on Android. *Network and Distributed Systems Symposium (NDSS)* (2019).

[24] Daniele Antonioli, Nils Ole Tippenhauer, and Kasper Rasmussen. 2020. BIAS: Bluetooth Impersonation AttackS. *Proceedings of the IEEE Symposium on Security and Privacy (S&P)* (2020).

[25] Daniele Antonioli, Nils Ole Tippenhauer, and Kasper Rasmussen. 2020. Key Negotiation Downgrade Attacks on Bluetooth and Bluetooth Low Energy. *ACM Transactions on Privacy and Security (TOPS)* (2020).

[26] Daniele Antonioli, Nils Ole Tippenhauer, and Kasper B Rasmussen. 2019. The KNOB is Broken: Exploiting Low Entropy in the Encryption Key Negotiation Of Bluetooth BR/EDR. *28th USENIX Security Symposium* (2019).

[27] Noah Apthorpe, Danny Yuxing Huang, Dillon Reisman, Arvind Narayanan, and Nick Feamster. 2019. Keeping the smart home private with smart (er) iot traffic shaping. *Proceedings on Privacy Enhancing Technologies (PoPETS)* (2019).

[28] Noah Apthorpe, Dillon Reisman, Srikanth Sundaresan, Arvind Narayanan, and Nick Feamster. 2017. Spying on the smart home: Privacy attacks and defenses on encrypted iot traffic. *arXiv preprint arXiv:1708.05044* (2017).

[29] Ludovic Barman, Italo Dacosta, Mahdi Zamani, Ennan Zhai, Apostolos Pyrgelis, Bryan Ford, Joan Feigenbaum, and Jean-Pierre Hubaux. 2020. PriFi: Low-Latency Anonymity for Organizational Networks. *Proceedings on Privacy Enhancing Technologies (PoPETS)* (2020).

[30] Ludovic Barman, Alexandre Dumur, Apostolos Pyrgelis, and Jean-Pierre Hubaux. 2021. Repository for "Every Byte Matters: Traffic Analysis of Bluetooth Wearable Devices". https://wearables.lbarman.ch. Accessed: 2021-05-01.

[31] Johannes K Becker, David Li, and David Starobinski. 2019. Tracking anonymized bluetooth devices. *Proceedings on Privacy Enhancing Technologies (PoPETS)* (2019).

[32] SIG Bluetooth. 2016. Bluetooth core specification v5.1. *Bluetooth Special Interest Group (SIG)* (2016).

[33] Xiang Cai, Rishab Nithyanand, and Rob Johnson. 2014. CS-BuFLO: A congestion sensitive website fingerprinting defense. *Proceedings of the 13th Workshop on Privacy in the Electronic Society (WPES)* (2014).

[34] Xiang Cai, Rishab Nithyanand, Tao Wang, Rob Johnson, and Ian Goldberg. 2014. A systematic approach to developing and evaluating website fingerprinting defenses. *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security (CCS)* (2014).

[35] Guillaume Celosia and Mathieu Cunche. 2019. Saving private addresses: an analysis of privacy issues in the bluetooth-low-energy advertising mechanism. *Proceedings of the 16th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous)* (2019).

[36] Guillaume Celosia and Mathieu Cunche. 2020. Discontinued Privacy: Personal Data Leaks in Apple Bluetooth-Low-Energy Continuity Protocols. *Proceedings on Privacy Enhancing Technologies (PoPETS)* (2020).

[37] Chen Chen, Daniele E Asoni, Adrian Perrig, David Barrera, George Danezis, and Carmela Troncoso. 2018. TARANET: Traffic-analysis resistant anonymity at the network layer. *2018 IEEE European Symposium on Security and Privacy (EuroS&P)* (2018).

[38] Jiska Classen and Matthias Hollick. 2019. Inside job: Diagnosing bluetooth lower layers using Off-the-shelf devices. *Proceedings of the 12th Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)* (2019).

[39] Jiska Classen, Daniel Wegemer, Paul Patras, Tom Spink, and Matthias Hollick. 2018. Anatomy of a vulnerable fitness tracking system: Dissecting the fitbit cloud, app, and firmware. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies (IMWUT)* (2018).

[40] Marco Cominelli, Francesco Gringoli, Paul Patras, Margus Lind, and Guevara Noubir. 2020. Even black cats cannot stay hidden in the dark: Full-band de-anonymization of bluetooth classic devices. *2020 IEEE Symposium on Security and Privacy (S&P)* (2020).

[41] Marco Cominelli, Paul Patras, and Francesco Gringoli. 2020. One GPU to Snoop Them All: a Full-Band Bluetooth Low Energy Sniffer. *2020 Mediterranean Communication and Computer Networking Conference (MedComNet)* (2020).

[42] Mauro Conti, Luigi Vincenzo Mancini, Riccardo Spolaor, and Nino Vincenzo Verde. 2015. Analyzing android encrypted network traffic to identify user actions. *IEEE Transactions on Information Forensics and Security (TIFS)* (2015).

[43] Henry Corrigan-Gibbs, Dan Boneh, and David Mazières. 2015. Riposte: An anonymous messaging system handling millions of users. *2015 IEEE Symposium on Security and Privacy (S&P)* (2015).

[44] George Danezis and Richard Clayton. 2007. Introducing Traffic Analysis. *Digital Privacy: Theory, Technologies, and Practices* (Dec. 2007).

[45] Aveek K Das, Parth H Pathak, Chen-Nee Chuah, and Prasant Mohapatra. 2016. Uncovering privacy leakage in ble network traffic of wearable fitness trackers. *Proceedings of the 17th International Workshop on Mobile Computing Systems and Applications* (2016).

[46] Debajyoti Das, Sebastian Meiser, Esfandiar Mohammadi, and Aniket Kate. 2018. Anonymity trilemma: Strong anonymity, low bandwidth overhead, low latency-choose two. *2018 IEEE Symposium on Security and Privacy (S&P)* (2018).

[47] Kushal Dave, Vasudeva Varma, et al. 2014. Computational advertising: Techniques for targeting relevant ads. *Foundations and Trends® in Information Retrieval* (2014).

[48] Kevin P Dyer, Scott E Coull, Thomas Ristenpart, and Thomas Shrimpton. 2012. Peek-a-boo, i still see you: Why efficient traffic analysis countermeasures fail. *2012 IEEE Symposium on Security and Privacy (S&P)* (2012).

[49] Jiajun Gong and Tao Wang. 2020. Zero-delay Lightweight Defenses against Website Fingerprinting. *29th USENIX Security Symposium* (2020).

[50] Keijo Haataja and Pekka Toivanen. 2008. Practical man-in-the-middle attacks against bluetooth secure simple pairing. *2008 4th International Conference on Wireless Communications, Networking and Mobile Computing (WiCom)* (2008).

[51] Keijo Haataja and Pekka Toivanen. 2010. Two practical man-in-the-middle attacks on bluetooth secure simple pairing and countermeasures. *IEEE Transactions on Wireless Communications* (2010).

[52] Keijo MJ Haataja and Konstantin Hypponen. 2008. Man-in-the-middle attacks on bluetooth: a comparative analysis, a novel attack, and countermeasures. *2008 3rd International Symposium on Communications, Control and Signal Processing* (2008).

[53] Jamie Hayes and George Danezis. 2016. k-fingerprinting: A robust scalable website fingerprinting technique. *25th USENIX Security Symposium* (2016).

[54] Dominik Herrmann, Rolf Wendolsky, and Hannes Federrath. 2009. Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier. *Proceedings of the 2009 ACM workshop on Cloud computing security* (2009).

[55] Andrew Hilts, Christopher Parsons, and Jeffrey Knockel. 2016. Every step you fake: A comparative analysis of fitness tracker privacy and security. *Open Effect Report* (2016).

[56] Jun Huang, Wahhab Albazrqaoe, and Guoliang Xing. 2014. Blueid: A practical system for bluetooth device identification. *IEEE Conference on Computer Communications (INFOCOM)* (2014).

[57] Marc Juarez, Mohsen Imani, Mike Perry, Claudia Diaz, and Matthew Wright. 2016. Toward an efficient website fingerprinting defense. *European Symposium on Research in Computer Security (ESORICS)* (2016).

[58] Aleksandra Korolova and Vinod Sharma. 2018. Cross-app tracking via nearby Bluetooth low energy devices. *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy (CODASPY)* (2018).

[59] Albert Kwon, David Lazar, Srinivas Devadas, and Bryan Ford. 2016. Riffle: An efficient communication system with strong anonymity. *Proceedings on Privacy Enhancing Technologies (PoPETS)* (2016).

[60] Marc Liberatore and Brian Neil Levine. 2006. Inferring the source of encrypted HTTP connections. *Proceedings of the 13th ACM SIGSAC Conference on Computer and Communications Security (CCS)* (2006).

[61] Xing Liu, Tianyu Chen, Feng Qian, Zhixiu Guo, Felix Xiaozhu Lin, Xiaofeng Wang, and Kai Chen. 2017. Characterizing smartwatch usage in the wild. In *Proceedings of the 15th International Conference on Mobile Systems, Applications, and Services (MobiSys).*

[62] Xiapu Luo, Peng Zhou, Edmond WW Chan, Wenke Lee, Rocky KC Chang, and Roberto Perdisci. 2011. HTTPOS: Sealing Information Leaks with Browser-side Obfuscation of Encrypted Flows. *Network and Distributed Systems Symposium (NDSS)* (2011).

[63] Dennis Mantz, Jiska Classen, Matthias Schulz, and Matthias Hollick. 2019. Internalblue-bluetooth binary patching and experimentation framework. *Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys)* (2019).

[64] Jeremy Martin, Douglas Alpuche, Kristina Bodeman, Lamont Brown, Ellis Fenske, Lucas Foppe, Travis Mayberry, Erik Rye, Brandon Sipes, and Sam Teplov. 2019. Handoff all your privacy–a review of apple's bluetooth low energy continuity protocol. *Proceedings on Privacy Enhancing Technologies (PoPETS)* (2019).

[65] Jonathan Muehlstein, Yehonatan Zion, Maor Bahumi, Itay Kirshenboim, Ran Dubin, Amit Dvir, and Ofir Pele. 2017. Analyzing HTTPS encrypted traffic to identify user's operating system, browser and application. *2017 14th IEEE Annual Consumer Communications & Networking Conference (CCNC)* (2017).

[66] Andriy Panchenko, Fabian Lanze, Jan Pennekamp, Thomas Engel, Andreas Zinnen, Martin Henze, and Klaus Wehrle. 2016. Website Fingerprinting at Internet Scale. *Network and Distributed Systems Symposium (NDSS)* (2016).

[67] Andriy Panchenko, Lukas Niessen, Andreas Zinnen, and Thomas Engel. 2011. Website fingerprinting in onion routing based anonymization networks. *Proceedings of the 10th annual ACM workshop on Privacy in the Electronic Society (WPES)* (2011).

[68] Jeffrey Pang, Ben Greenstein, Ramakrishna Gummadi, Srinivasan Seshan, and David Wetherall. 2007. 802.11 User Fingerprinting. *ACM International Conference on Mobile Computing and Networking (MobiCom)* (2007).

[69] Claudia Perlich, Brian Dalessandro, Troy Raeder, Ori Stitelman, and Foster Provost. 2014. Machine learning for targeted display advertising: Transfer learning in action. *Machine learning* (2014).

[70] Raphael C-W Phan and Patrick Mingard. 2012. Analyzing the secure simple pairing in Bluetooth v4. 0. *Wireless Personal Communications* (2012).

[71] Mohammad Saidur Rahman, Payap Sirinam, Nate Mathews, Kantha Girish Gangadhara, and Matthew Wright. 2020. Tik-Tok: The utility of packet timing in website fingerprinting attacks. *Proceedings on Privacy Enhancing Technologies (PoPETS)* (2020).

[72] Andrew Reed and Benjamin Klimkowski. 2016. Leaky streams: Identifying variable bitrate DASH videos streamed over encrypted 802.11n connections. *IEEE Consumer Communications & Networking Conference (CCNC)* (2016).

[73] Andrew Reed and Michael Kranch. 2017. Identifying HTTPS-protected Netflix videos in real-time. *Proceedings of the Seventh ACM Conference on Data and Application Security and Privacy (CODASPY)* (2017).

[74] Jan Ruge, Jiska Classen, Francesco Gringoli, and Matthias Hollick. 2020. Frankenstein: Advanced Wireless Fuzzing to Exploit New Bluetooth Escalation Targets. *29th USENIX Security Symposium* (2020).

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

[76] Brendan Saltaformaggio, Hongjun Choi, Kristen Johnson, Yonghwi Kwon, Qi Zhang, Xiangyu Zhang, Dongyan Xu, and John Qian. 2016. Eavesdropping on fine-grained user activities within smartphone apps over encrypted network traffic. *10th USENIX Workshop on Offensive Technologies (WOOT)* (2016).

[77] Roei Schuster, Vitaly Shmatikov, and Eran Tromer. 2017. Beauty and the Burst: Remote Identification of Encrypted Video Streams. *29th USENIX Security Symposium* (2017).

[78] Payap Sirinam, Mohsen Imani, Marc Juarez, and Matthew Wright. 2018. Deep fingerprinting: Undermining website fingerprinting defenses with deep learning. *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS)* (2018).

[79] Payap Sirinam, Nate Mathews, Mohammad Saidur Rahman, and Matthew Wright. 2019. Triplet Fingerprinting: More Practical and Portable Website Fingerprinting with N-shot Learning. *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS)* (2019).

[80] Dominic Spill and Andrea Bittau. 2007. BlueSniff: Eve Meets Alice and Bluetooth. *1st USENIX Workshop on Offensive Technologies (WOOT)* (2007).

[81] Vijay Srinivasan, John Stankovic, and Kamin Whitehouse. 2008. Protecting your daily in-home activity information from a wireless snooping attack. *Proceedings of the 10th international conference on Ubiquitous computing (UbiComp)* (2008).

[82] Ahmad Ali Tabassam and Stefan Heiss. 2008. Bluetooth clock recovery and hop sequence synchronization using software defined radios. *2008 IEEE Region 5 Conference* (2008).

[83] Vincent F Taylor, Riccardo Spolaor, Mauro Conti, and Ivan Martinez. 2017. Robust smartphone app identification via encrypted network traffic analysis. *IEEE Transactions on Information Forensics and Security (TIFS)* (2017).

[84] Vincent F Taylor, Riccardo Spolaor, Mauro Conti, and Ivan Martinovic. 2016. Appscanner: Automatic fingerprinting of smartphone apps from encrypted network traffic. *2016 IEEE European Symposium on Security and Privacy (EuroS&P)* (2016).

[85] Rahmadi Trimananda, Janus Varmarken, Athina Markopoulou, and Brian Demsky. 2020. Packet-level signatures for smart home devices. *Network and Distributed Systems Symposium (NDSS)* (2020).

[86] Jiliang Wang, Feng Hu, Ye Zhou, Yunhao Liu, Hanyi Zhang, and Zhe Liu. 2020. BlueDoor: breaking the secure information flow via BLE vulnerability. *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services (MobiSys)* (2020).

[87] Tao Wang, Xiang Cai, Rishab Nithyanand, Rob Johnson, and Ian Goldberg. 2014. Effective attacks and provable defenses for website fingerprinting. *23rd USENIX Security Symposium* (2014).

[88] Tao Wang and Ian Goldberg. 2013. Improved Website Fingerprinting on Tor. *Proceedings of the 12th annual ACM workshop on Privacy in the Electronic Society (WPES)* (2013).

[89] Tao Wang and Ian Goldberg. 2016. On realistically attacking Tor with website fingerprinting. *Proceedings on Privacy Enhancing Technologies (PoPETS)* (2016).

[90] Tao Wang and Ian Goldberg. 2017. Walkie-talkie: An efficient defense against passive website fingerprinting attacks. *26th USENIX Security Symposium* (2017).

[91] Ford-Long Wong, Frank Stajano, and Jolyon Clulow. 2005. Repairing the Bluetooth pairing protocol. *International Workshop on Security Protocols* (2005).

[92] Charles V Wright, Scott E Coull, and Fabian Monrose. 2009. Traffic Morphing: An Efficient Defense Against Statistical Traffic Analysis. *Network and Distributed Systems Symposium (NDSS)* (2009).

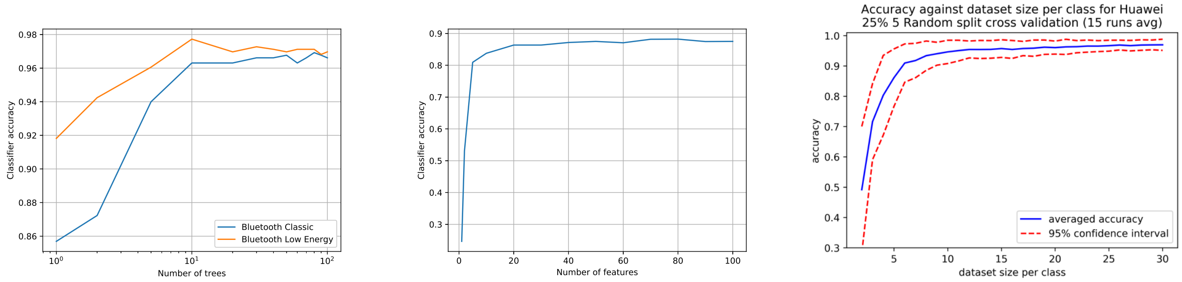
[93] Jianliang Wu, Yuhong Nan, Vireshwar Kumar, Dave Jing Tian, Antonio Bianchi, Mathias Payer, and Dongyan Xu. 2020. BLESA: Spoofing Attacks against Reconnections in Bluetooth Low Energy. *14th USENIX Workshop on Offensive Technologies (WOOT)* (2020).

[94] Fan Zhang, Wenbo He, Xue Liu, and Patrick G. Bridges Bridges. 2011. Inferring Users' Online Activities Through Traffic Analysis. *Proceedings of the 4th Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)* (June 2011).

[95] Yue Zhang, Jian Weng, Rajib Dey, Yier Jin, Zhiqiang Lin, and Xinwen Fu. 2020. Breaking secure pairing of bluetooth low energy using downgrade attacks. *29th USENIX Security Symposium* (2020).

[96] Chaoshun Zuo, Haohuang Wen, Zhiqiang Lin, and Yinqian Zhang. 2019. Automatic fingerprinting of vulnerable ble iot devices with static uuids from mobile apps. *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS)* (2019).

A ADDITIONAL FIGURES

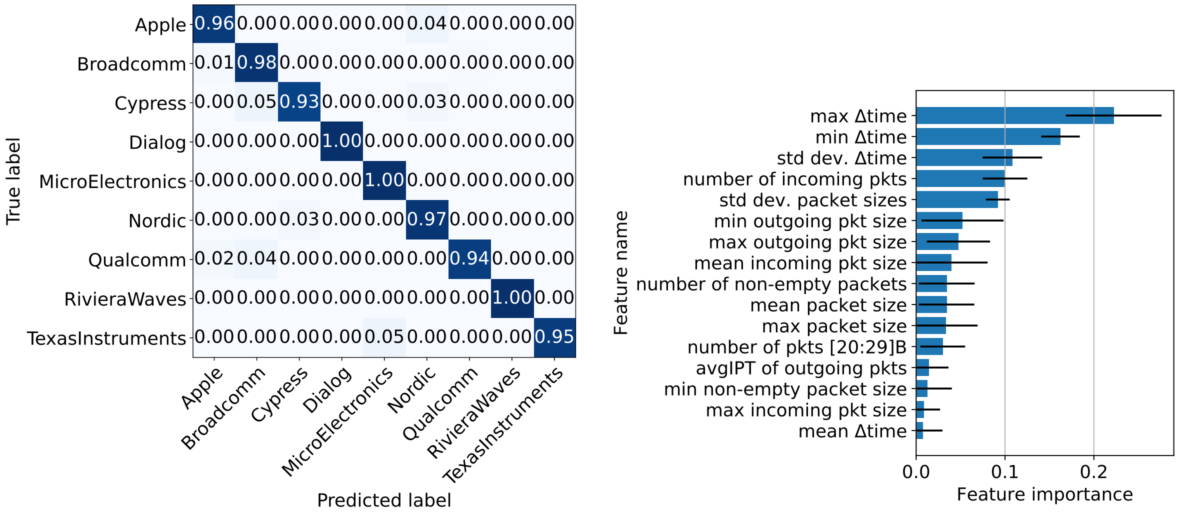


(a) Number of trees, device identification (§5).

(b) Number of features kept by the Recursive Feature Elimination (RFE), device identification (§5).

(c) Number of samples per class, application identification ("deep" experiment, §6.2).

Fig. 11. Choice of parameters versus mean classifier accuracy.



True label	Apple	Broadcomm	Cypress	Dialog	MicroElectronics	Nordic	Qualcomm	RivieraWaves	TexasInstruments
Apple	0.96	0.00	0.00	0.00	0.00	0.04	0.00	0.00	0.00
Broadcomm	0.01	0.98	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Cypress	0.00	0.05	0.93	0.00	0.00	0.03	0.00	0.00	0.00
Dialog	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00
MicroElectronics	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00
Nordic	0.00	0.00	0.03	0.00	0.00	0.97	0.00	0.00	0.00
Qualcomm	0.02	0.04	0.00	0.00	0.00	0.00	0.94	0.00	0.00
RivieraWaves	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00
TexasInstruments	0.00	0.00	0.00	0.00	0.05	0.00	0.00	0.00	0.95

(a) Normalized confusion matrix per true label.

(b) Feature importance.

Fig. 12. Chipset identification (Classic & LE)

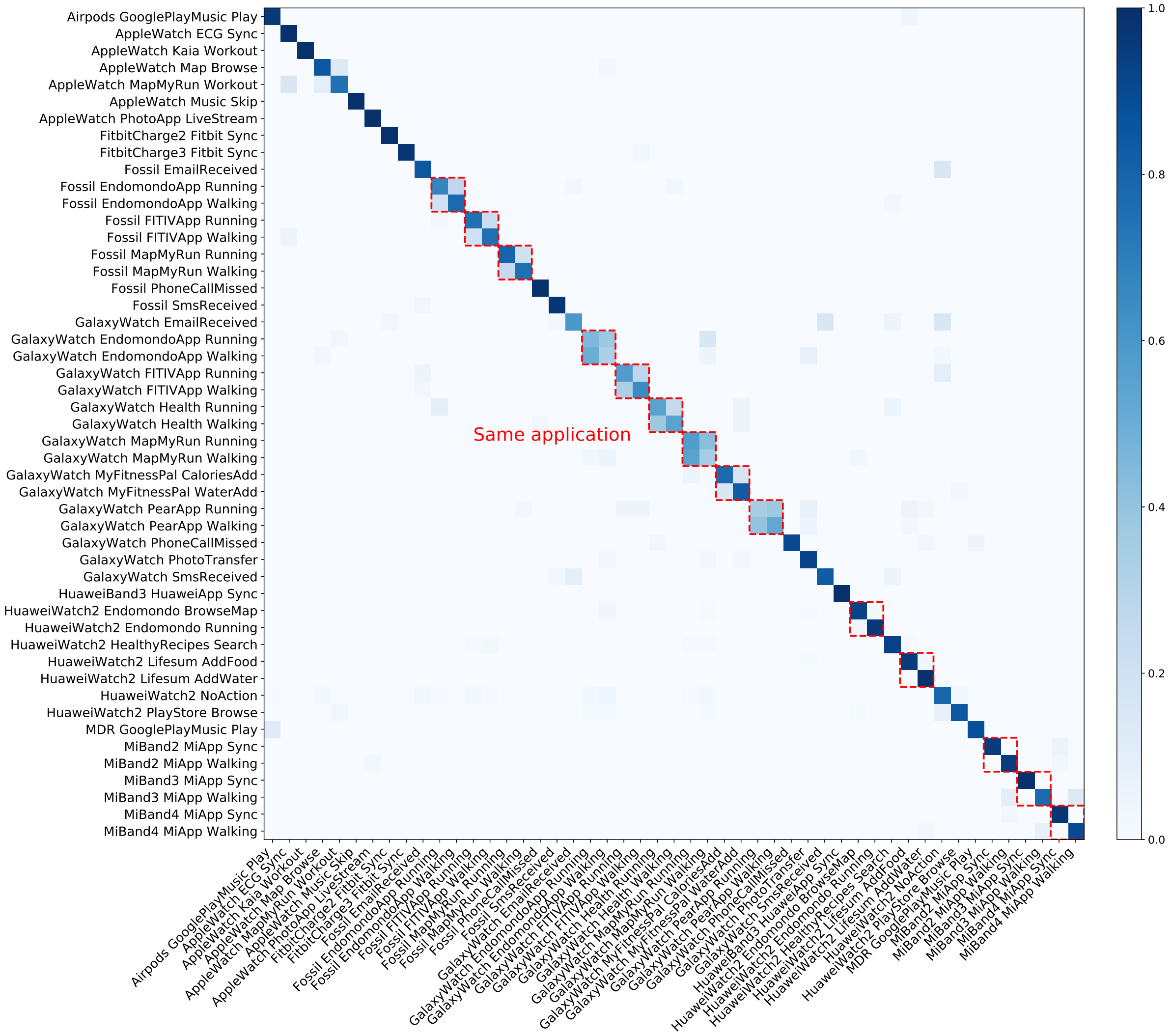
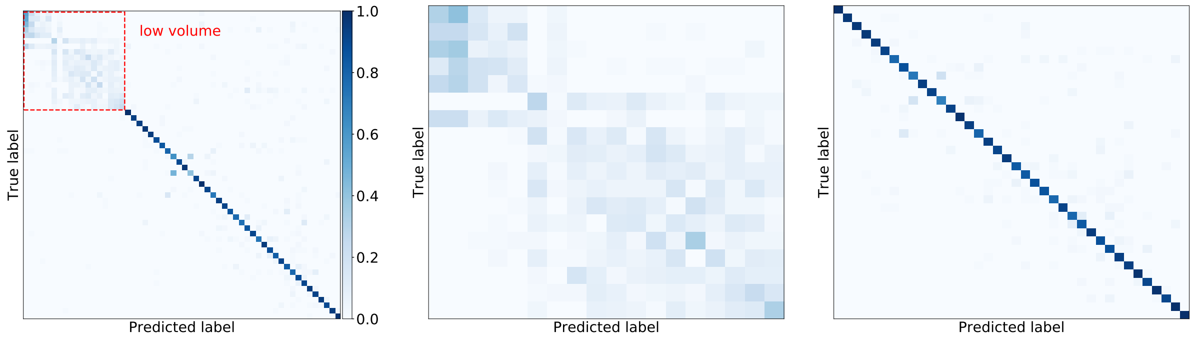


Fig. 13. Confusion matrix for human-triggered actions. Red dashed squares regroup the two actions “start a walking workout” and “start a running workout” within the same application.

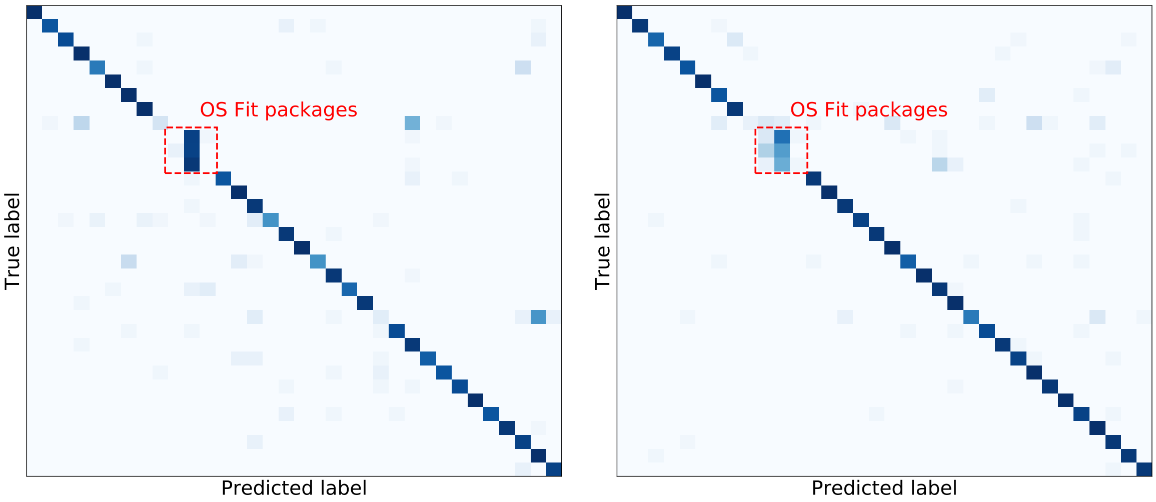


(a) Normalized confusion matrix for 56 apps, Wear OS. Apps are sorted by increasing median transmitted volume.

(b) Normalized confusion matrix for the 18 "low-volume" apps, Wear OS. The mean accuracy is 17%.

(c) Normalized confusion matrix for 38 high-volume apps, Wear OS. The mean accuracy is 90%.

Fig. 14. Normalized confusion matrices per true label for recognizing smartwatch application openings.



(a) Train on Huawei-Pixel. Test on Fossil-Nexus.

(b) Train on Fossil-Nexus. Test on Huawei-Pixel.

Fig. 15. Transferability experiment for the application identification ("deep" experiment).

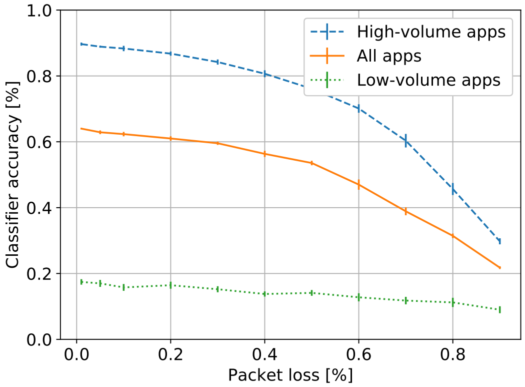


Fig. 16. Packet loss rate versus classifier accuracy for application identification ("deep" experiment).

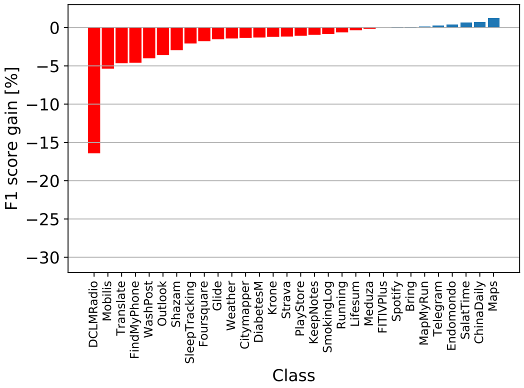
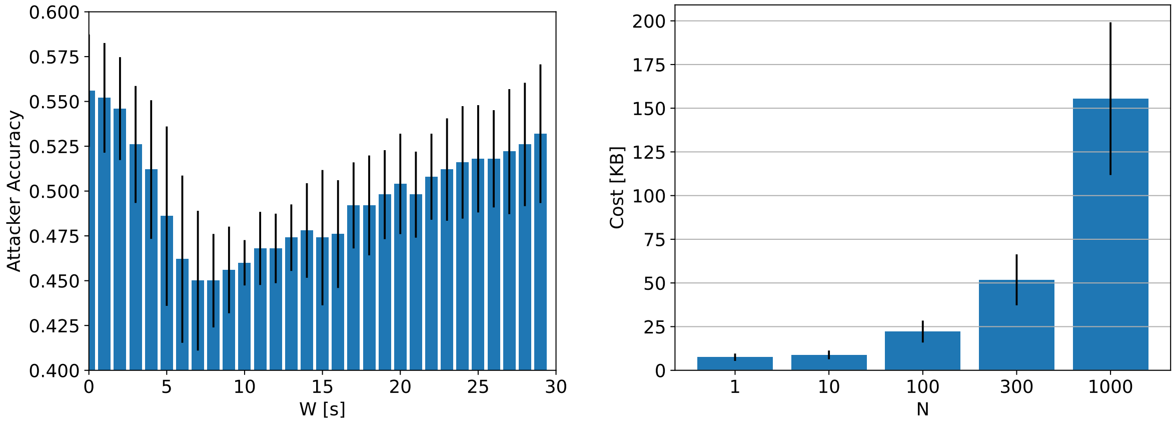


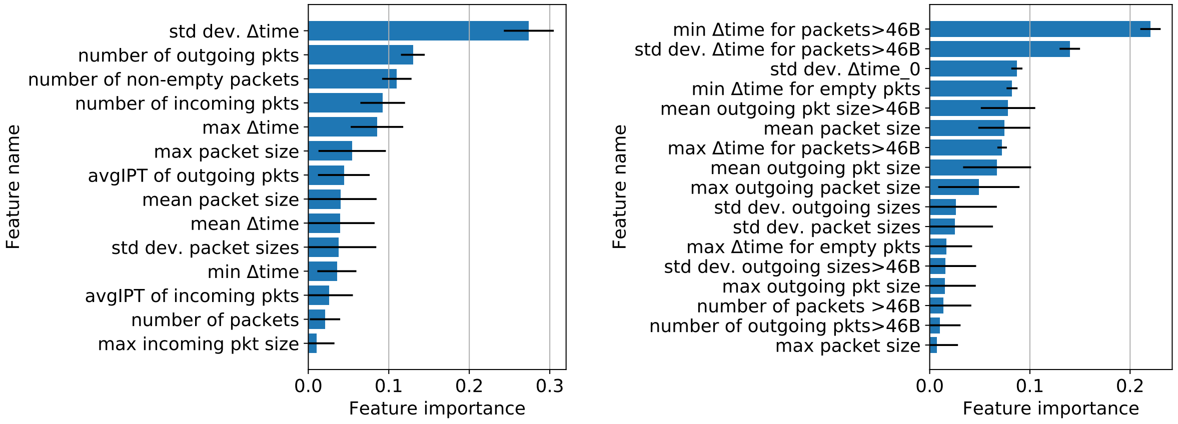
Fig. 17. Evolution of the F1 score per class, averaged over 29 days, training over the initial three days. See Figure 10(b) for the case of training on day 0 only.



(a) Mean W of the Rayleigth distribution with respect to the attacker accuracy (lower is better).

(b) Number of dummies N with respect to the cost of the defense. The cost is averaged per sample.

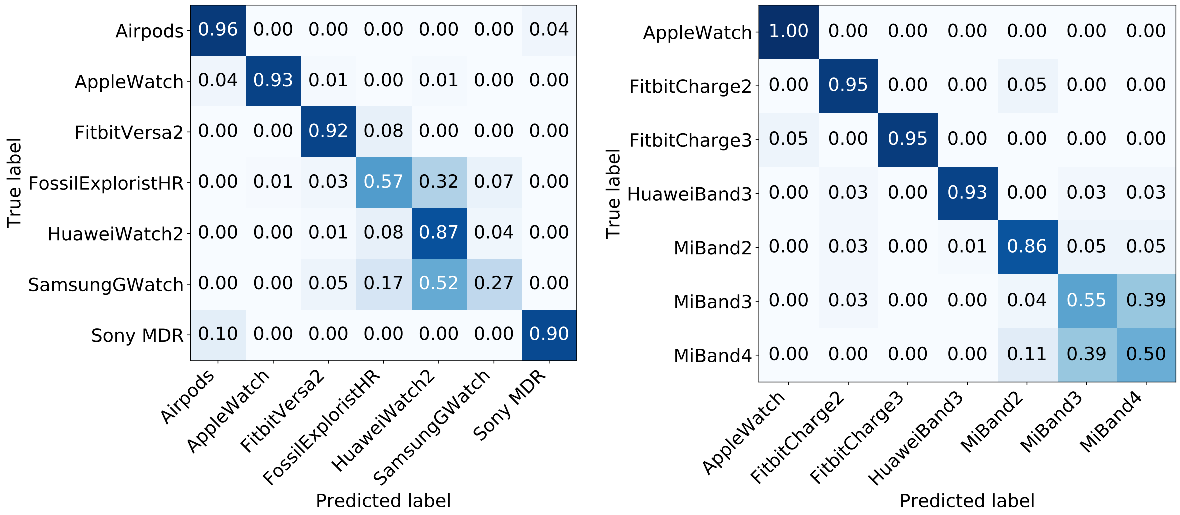
Fig. 18. Parameter choices for add_dummies, illustrated here with the samples for application identification ("deep" experiment). We select $W = 6$s and $N = 300$.



(a) Feature importance when attacking pad-defended traces, device identification, Bluetooth LE.

(b) Feature importance for action identification against delay_group-defended traces, DiabetesM.

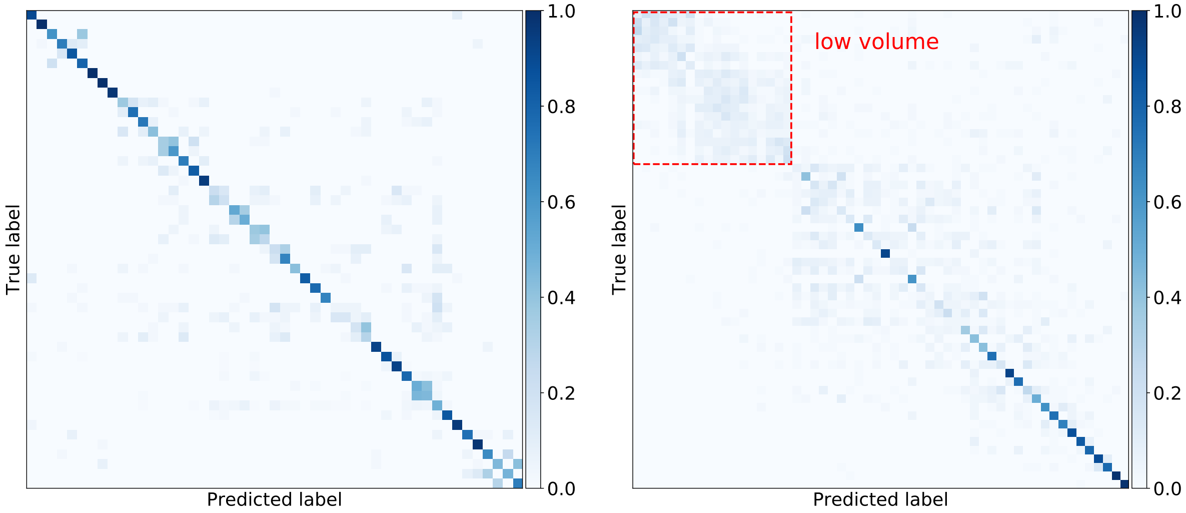
Fig. 19. Feature importance when attacking defended traces, two scenarios.



(a) Bluetooth Classic

(b) Bluetooth Low Energy

Fig. 20. Normalized confusion matrices per true label, device identification against add_dummies-defended traces.



(a) Action identification ("wide" experiment on all wearables).

(b) Application identification ("deep" experiment).

Fig. 21. Normalized confusion matrices per true label. Action and application identification against add_dummies-defended traces. The matrices are sorted by increasing median transmitted volume.

B TABLES

Classifier performance for Device identification.

Table A1. Bluetooth Classic.

Label	Precision	Recall	F1-score
Airpods	0.83	0.9	0.87
AppleWatch	0.98	0.93	0.96
FitbitVersa2	1.0	1.0	1.0
FossilExploristHR	0.96	0.98	0.97
HuaweiWatch2	0.98	0.98	0.98
SamsungGWatch	1.0	0.98	0.99
Sony MDR	0.9	0.86	0.88
Average	0.96	0.96	0.96

Table A2. Bluetooth LE.

Label	Precision	Recall	F1-score
AppleWatch	0.99	1.0	1.0
FitbitCharge2	1.0	1.0	1.0
FitbitCharge3	1.0	0.95	0.97
HuaweiBand3	1.0	1.0	1.0
MiBand2	0.93	0.94	0.93
MiBand3	0.88	0.98	0.92
MiBand4	0.99	0.86	0.92
Average	0.97	0.97	0.97

Classifier performances.

Table A3. Chipset identification.

Label	Precision	Recall	F1-score
Apple	0.9	0.94	0.92
Broadcomm	0.95	0.98	0.96
Cypress	0.98	0.98	0.98
Dialog	0.98	0.98	0.98
MicroElectronics	1.0	1.0	1.0
Qualcomm	0.97	0.88	0.92
RivieraWaves	0.98	1.0	0.99
Average	0.96	0.96	0.96

Table A4. Action identification within `DiabetesM`.

Label	Precision	Recall	F1-score
Add Calorie	0.44	0.46	0.45
Add Carbs	0.87	0.9	0.89
Add Fat	0.77	0.81	0.79
Add Glucose	0.85	0.91	0.88
Add Insulin	0.9	0.95	0.92
Add Proteins	0.37	0.28	0.32
Average	0.7	0.7	0.7

Table A5. 34 "high-volume" applications common between the two pairs of device `Huawei Watch 2 - Pixel 2` and `Fossil Q Explorist HR - Nexus 5`.

`Bring`, `Calm`, `ChinaDaily`, `Citymapper`, `DCLMRadio`, `DiabetesM`, `Endomondo`, `FITIVPlus`, `FindMyPhone`, `Fit`, `FitBreathe`, `FitWorkout`, `FoursquareCityGuide`, `Glide`, `KeepNotes`, `Krone`, `Lifesum`, `MapMyRun`, `Maps`, `Meduza`, `Mobills`, `Outlook`, `PlayStore`, `Running`, `SalatTime`, `Shazam`, `SleepTracking`, `SmokingLog`, `Spotify`, `Strava`, `Telegram`, `Translate`, `WashPost`, `Weather`

Table A6. Applications and Actions used for the long-run captures.

Applications	`AppInTheAir`, `Bring`, `Calm`, `ChinaDaily`, `Citymapper`, `DCLMRadio`, `DiabetesM`, `Endomondo`, `FITIVPlus`, `FindMyPhone`, `FoursquareCityGuide`, `Glide`, `KeepNotes`, `Krone`, `Lifesum`, `MapMyRun`, `Maps`, `Meduza`, `Mobills`, `Outlook`, `PlayStore`, `Qardio`, `Running`, `SalatTime`, `Shazam`, `SleepTracking`, `SmokingLog`, `Spotify`, `Strava`, `Telegram`, `Translate`, `WashPost`, `Weather`.
Actions	`DiabetesM_AddCalorie`, `DiabetesM_AddCarbs`, `DiabetesM_AddFat`, `DiabetesM_AddGlucose`, `DiabetesM_AddInsulin`, `DiabetesM_AddProteins`, `Endomondo_BrowseMap`, `Endomondo_Running`, `FoursquareCityGuide_Coffees`, `FoursquareCityGuide_Leisure`, `FoursquareCityGuide_NightLife`, `FoursquareCityGuide_Restaurants`, `FoursquareCityGuide_Shopping`, `HealthyRecipes_SearchRecipe`, `Lifesum_AddFood`, `Lifesum_AddWater`, `PlayStore_Browse`.

Classifier performance for Action identification, "Wide" experiment on all wearables.

Label	**Precision**	**Recall**	**F1-score**
Airpods GooglePlayMusic Play	0.87	0.96	0.91
AppleWatch ECG Sync	0.83	1.0	0.91
AppleWatch Kaia Workout	1.0	1.0	1.0
AppleWatch Map Browse	0.83	0.85	0.84
AppleWatch MapMyRun Workout	0.79	0.75	0.77
AppleWatch Music Skip	1.0	1.0	1.0
AppleWatch PhotoApp LiveStream	0.98	1.0	0.99
FitbitCharge2 Fitbit Sync	0.98	1.0	0.99
FitbitCharge3 Fitbit Sync	1.0	0.98	0.99
FossilExploristHR EndomondoApp Running	0.66	0.68	0.67
FossilExploristHR EndomondoApp Walking	0.74	0.78	0.76
FossilExploristHR FITIVApp Running	0.71	0.75	0.73
FossilExploristHR FITIVApp Walking	0.71	0.75	0.73
FossilExploristHR MapMyRun Running	0.76	0.8	0.78
FossilExploristHR MapMyRun Walking	0.77	0.75	0.76
FossilExploristHR NoApp EmailReceived	0.83	0.85	0.84
FossilExploristHR NoApp PhoneCallMissed	0.98	1.0	0.99
FossilExploristHR NoApp SmsReceived	0.95	0.98	0.96
GalaxyWatch EndomondoApp Running	0.42	0.45	0.43
GalaxyWatch EndomondoApp Walking	0.32	0.32	0.32
GalaxyWatch FITIVApp Running	0.61	0.57	0.59
GalaxyWatch FITIVApp Walking	0.65	0.65	0.65
GalaxyWatch MapMyRun Running	0.47	0.57	0.52
GalaxyWatch MapMyRun Walking	0.31	0.35	0.33
GalaxyWatch MyFitnessPalApp CaloriesAdd	0.84	0.78	0.81
GalaxyWatch MyFitnessPalApp WaterAdd	0.73	0.82	0.78
GalaxyWatch NoApp EmailReceived	0.83	0.6	0.7
GalaxyWatch NoApp PhoneCallMissed	1.0	0.9	0.95
GalaxyWatch NoApp PhotoTransfer	0.79	0.92	0.85
GalaxyWatch NoApp SmsReceived	0.85	0.82	0.84
GalaxyWatch PearApp Running	0.47	0.35	0.4
GalaxyWatch PearApp Walking	0.58	0.52	0.55
GalaxyWatch SamsungHealthApp Running	0.58	0.55	0.56
GalaxyWatch SamsungHealthApp Walking	0.67	0.55	0.6
HuaweiBand3 HuaweiApp Sync	1.0	1.0	1.0
HuaweiWatch2 Endomondo BrowseMap	0.94	0.92	0.93
HuaweiWatch2 Endomondo Running	0.98	0.97	0.97
HuaweiWatch2 HealthyRecipes SearchRecipe	0.92	0.93	0.93
HuaweiWatch2 Lifesum AddFood	0.93	0.95	0.94
HuaweiWatch2 Lifesum AddWater	0.93	1.0	0.96
HuaweiWatch2 NoApp NoAction	0.81	0.79	0.8
HuaweiWatch2 PlayStore Browse	0.94	0.85	0.89
MDR GooglePlayMusic Play	0.96	0.88	0.92
MiBand2 MiApp Sync	1.0	0.95	0.97
MiBand2 MiApp Walking	0.88	0.95	0.92
MiBand3 MiApp Sync	1.0	1.0	1.0
MiBand3 MiApp Walking	0.91	0.78	0.84
MiBand4 MiApp Sync	0.93	0.98	0.95
MiBand4 MiApp Walking	0.88	0.9	0.89
Average	0.82	0.82	0.82

Table A7. Details of transmitted volumes for the 18 "low-volume" apps over 40 recorded samples. NoApp corresponds to OS communications.

App	Median [B]	Std dev. [B]
Reminders	0.0	23660.5
Battery	0.0	239.6
DuaKhatqmAlQuran	11.0	43168.3
WearCasts	37.0	398149.0
DailyTracking	44.0	326.9
ASB	75.0	3511.6
NoApp	96.5	3949.5
HeartRate	104.0	21824.4
Workout	119.0	5108.4
AthkarOfPrayer	120.0	4835.5
Alarm	122.5	8891.4
GooglePay	127.0	2762.1
Flashlight	152.5	4176.7
Phone	154.5	2319.2
PlayMusic	156.0	16294.5
HealthyRecipes	167.5	3104.7
Sleep	171.5	14460.4
Medisafe	194.0	8802.6

Table A8. Details of transmitted volumes for the 38 "high-volume" apps over 40 samples.

App	Median [KB]	Std dev. [KB]
SalatTime	0.6	5.5
MapMyFitness	0.6	2.6
Citymapper	1.1	3.4
Calm	1.2	8.3
Outlook	1.4	3.4
DiabetesM	1.4	2.5
SmokingLog	1.5	46.7
MapMyRun	1.8	8.1
SleepTracking	1.9	4.4
Mobills	2.1	1.5
Fit	2.1	8.4
Weather	2.6	9.4
Running	2.9	8.0
FitWorkout	3.3	144.6
FitBreathe	3.3	3.4
FoursquareCityGuide	3.8	7.0
Glide	4.8	5.2
Translate	5.8	5.9
Shazam	6.2	41.9
Qardio	6.5	7.1
Krone	6.8	13.2
KeepNotes	7.0	4.8
FindMyPhone	8.0	12.2
Telegram	8.7	3.7
Strava	10.6	4.7
DCLMRadio	16.6	15.6
Lifesum	17.8	11.0
Endomondo	17.9	9.7
PlayStore	18.5	24.3
Maps	18.7	62.8
AppInTheAir	26.4	14.4
Bring	34.5	7.9
Spotify	38.8	9.8
Meduza	40.9	13.9
FITIVPlus	48.9	11.7
ChinaDaily	55.1	8.3
WashPost	120.1	14.2
Camera	598.0	141.5

Classifier performance for App identification, Huawei Watch.

Table A9. 18 "low-volume" applications.

Label	**Precision**	**Recall**	**F1-score**
Battery	0.21	0.31	0.25
Reminders	0.14	0.25	0.18
DuaKhatqmAlQuran	0.08	0.06	0.07
WearCasts	0.26	0.18	0.21
DailyTracking	0.2	0.15	0.17
ASB	0.25	0.28	0.26
NoApp	0.06	0.02	0.04
HeartRate	0.17	0.15	0.16
Workout	0.11	0.1	0.11
AthkarOfPrayer	0.09	0.09	0.09
Alarm	0.06	0.05	0.05
GooglePay	0.08	0.09	0.08
Flashlight	0.13	0.14	0.13
Phone	0.34	0.34	0.34
PlayMusic	0.2	0.2	0.2
HealthyRecipes	0.11	0.12	0.12
Sleep	0.21	0.24	0.22
Medisafe	0.32	0.32	0.32
Average	0.17	0.17	0.17

Table A10. 38 "high-volume" applications

Label	**Precision**	**Recall**	**F1-score**
SalatTime	1.0	1.0	1.0
MapMyFitness	1.0	0.96	0.98
Calm	0.94	0.96	0.95
Citymapper	0.94	0.96	0.95
DiabetesM	0.94	0.95	0.94
Outlook	0.96	0.92	0.94
SmokingLog	0.91	0.79	0.85
Fit	0.82	0.88	0.85
Running	0.78	0.72	0.75
MapMyRun	1.0	0.94	0.97
SleepTracking	0.99	0.92	0.95
Weather	0.76	0.69	0.72
Mobills	0.95	0.92	0.94
FitBreathe	0.96	0.99	0.98
FoursquareCityGuide	0.91	0.94	0.93
FitWorkout	0.84	0.8	0.82
Glide	0.88	0.94	0.91
Translate	0.94	0.91	0.92
Qardio	0.95	0.95	0.95
Krone	0.91	0.84	0.87
FindMyPhone	0.81	0.84	0.82
KeepNotes	0.95	0.88	0.91
Shazam	0.95	0.86	0.9
Strava	0.86	0.79	0.82
Telegram	0.95	0.92	0.94
Maps	0.74	0.79	0.76
Endomondo	0.78	0.84	0.81
DCLMRadio	0.96	0.95	0.96
Lifesum	0.8	0.88	0.84
PlayStore	0.8	0.88	0.83
AppInTheAir	0.82	0.94	0.88
Bring	0.82	0.94	0.88
Spotify	0.96	0.99	0.98
Meduza	0.9	0.91	0.91
FITIVPlus	0.99	0.99	0.99
ChinaDaily	0.96	0.91	0.94
WashPost	0.9	1.0	0.95
Camera	1.0	1.0	1.0
Average	0.9	0.9	0.9

Classifier performance when transferring the model between pairs of devices.

Table A11. Train on Huawei-Pixel. Test on Fossil-Nexus.

Label	**Precision**	**Recall**	**F1-score**
Bring	1.0	1.0	1.0
Calm	0.96	0.86	0.91
ChinaDaily	0.96	0.89	0.93
Citymapper	0.74	1.0	0.85
DCLMRadio	0.91	0.71	0.8
DiabetesM	0.97	1.0	0.98
Endomondo	0.78	1.0	0.88
FITIVPlus	0.88	1.0	0.93
FindMyPhone	0.71	0.17	0.28
Fit	0.0	0.0	0.0
FitBreathe	0.3	0.93	0.46
FitWorkout	0.0	0.0	0.0
FoursquareCityGuide	1.0	0.86	0.92
Glide	0.85	1.0	0.92
KeepNotes	0.71	0.96	0.82
Krone	1.0	0.62	0.77
Lifesum	0.84	0.96	0.9
MapMyRun	1.0	1.0	1.0
Maps	0.95	0.62	0.75
Meduza	0.85	0.97	0.9
Mobills	1.0	0.79	0.88
Outlook	1.0	0.96	0.98
PlayStore	0.33	0.11	0.16
Running	0.96	0.89	0.93
SalatTime	0.57	0.96	0.72
Shazam	1.0	0.82	0.9
SleepTracking	0.96	0.86	0.91
SmokingLog	0.96	0.89	0.93
Spotify	1.0	1.0	1.0
Strava	1.0	0.86	0.92
Telegram	1.0	0.97	0.98
Translate	0.72	0.93	0.81
WashPost	0.57	1.0	0.73
Weather	0.93	0.93	0.93
Average	0.81	0.81	0.81

Table A12. Train on Fossil-Nexus. Test on Huawei-Pixel.

Label	**Precision**	**Recall**	**F1-score**
Bring	1.0	1.0	1.0
Calm	1.0	0.97	0.98
ChinaDaily	0.92	0.79	0.85
Citymapper	1.0	0.93	0.96
DCLMRadio	0.93	0.86	0.89
DiabetesM	1.0	1.0	1.0
Endomondo	0.83	0.86	0.84
FITIVPlus	0.87	0.96	0.92
FindMyPhone	0.67	0.07	0.13
Fit	0.21	0.14	0.17
FitBreathe	0.3	0.57	0.39
FitWorkout	0.5	0.04	0.07
FoursquareCityGuide	0.93	0.96	0.95
Glide	1.0	1.0	1.0
KeepNotes	0.93	0.96	0.95
Krone	0.96	0.93	0.95
Lifesum	0.96	0.96	0.96
MapMyRun	0.88	1.0	0.93
Maps	0.92	0.82	0.87
Meduza	1.0	1.0	1.0
Mobills	0.74	0.97	0.84
Outlook	0.85	1.0	0.92
PlayStore	0.95	0.71	0.82
Running	0.86	0.89	0.88
SalatTime	0.96	0.96	0.96
Shazam	0.9	0.93	0.91
SleepTracking	0.78	1.0	0.88
SmokingLog	0.97	0.97	0.97
Spotify	1.0	1.0	1.0
Strava	0.81	0.93	0.87
Telegram	0.76	1.0	0.86
Translate	0.8	0.97	0.88
WashPost	0.93	0.96	0.95
Weather	0.96	0.96	0.96
Average	0.86	0.86	0.86