Recurring Verification of Interaction Authenticity Within Bluetooth Networks

Travis Peters
Gianforte School of Computing,
Montana State University, USA

Timothy J. Pierson
Department of Computer Science,
Dartmouth College, USA

Sougata Sen
Department of Computer Science and Information Systems,
BITS Pilani, Goa Campus, India

José Camacho
Department of Signal Theory, Networking and Communications,
University of Granada, Spain

David Kotz
Department of Computer Science,
Dartmouth College, USA

ABSTRACT

Although *user* authentication has been well explored, *device-to-device* authentication – specifically in Bluetooth networks – has not seen the same attention. We propose *Verification of Interaction Authenticity (VIA)* – a recurring authentication scheme based on evaluating characteristics of the communications (*interactions*) between devices. We adapt techniques from wireless traffic analysis and intrusion-detection systems to develop behavioral models that capture typical, authentic device interactions (*behavior*); these models enable recurring verification of device behavior. To evaluate our approach we produced a new dataset consisting of more than 300 Bluetooth network traces collected from 20 Bluetooth-enabled smart-health and smart-home devices. In our evaluation, we found that devices can be correctly verified at a variety of granularities, achieving an F1-score of 0.86 or better in most cases.

CCS CONCEPTS

• **Security and privacy → Mobile and wireless security**; **Intrusion detection systems**; **Multi-factor authentication**.

KEYWORDS

Traffic Analysis, Behavioral Analysis, Bluetooth, Verification

ACM Reference Format:
Travis Peters, Timothy J. Pierson, Sougata Sen, José Camacho, and David Kotz. 2021. Recurring Verification of Interaction Authenticity Within Bluetooth Networks. In *Conference on Security and Privacy in Wireless and Mobile Networks (WiSec '21), June 28–July 2, 2021, Abu Dhabi, United Arab Emirates.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3448300.3468287

WiSec '21, June 28–July 2, 2021, Abu Dhabi, United Arab Emirates

ACM ISBN 978-1-4503-8349-3/21/06…$15.00
https://doi.org/10.1145/3448300.3468287

1 INTRODUCTION

Authentication is a critical security mechanism in many applications today. The most common forms of authentication are passwords, potentially used in combination with a second factor such as a hardware token or mobile app (i.e., two-factor authentication). These approaches, however, emphasize a *one-time, initial authentication*. After initial authentication, authenticated entities typically remain authenticated until an explicit *deauthentication* action is taken, or the authenticated session expires. Unfortunately, explicit deauthentication happens rarely, if ever.

Recent work [38, 51] has explored how to provide passive, continuous authentication and/or automatic deauthentication by correlating user movements and inputs with actions observed in an application (e.g., a web browser). This issue, however, goes beyond user authentication. Consider devices that pair via Bluetooth, for example, which commonly follow the pattern of *pair once, trust indefinitely*. After two devices connect, those devices are "bonded" together until a user explicitly removes the bond. Thus, this bond is likely to remain intact as long as the devices exist, or until they transfer ownership (e.g., are sold or lost).

Indefinitely trusting devices has become increasingly problematic in light of the increased adoption of IoT devices coupled with incessant reports of the inadequacy of their security [19, 43]. The reality of ubiquitous connectivity and frequent mobility gives rise to a myriad of opportunities for devices to be compromised. Thus, we argue that one-time, single-factor, device-to-device authentication (i.e., an initial pairing) is not enough, and that there must exist some mechanism to frequently (re-)verify the authenticity of devices and their connections.

In this paper we propose a device-to-device recurring authentication scheme – *Verification of Interaction Authenticity* (VIA) – that is based on evaluating characteristics of the communications (*interactions*) between devices. In the context of IoT and WPANs, VIA interposes on the communication channels between devices and their corresponding companion application. VIA extracts features from these interactions and compares them to an appropriate *verification model* to verify whether ongoing interactions are consistent with this known model. The devices are permitted to interact so long as the interactions remain consistent with previously-learned models that represent typical, *authentic* interactions.

To demonstrate the efficacy of VIA, our work primarily aims to address the following **research question:** *Can we reliably verify whether a device's behavior is consistent with a previously-learned, authentic behavioral model?* Here, we postulate that a device's interactions serve as a useful measure of the device's behavior. VIA relies on this notion of behavior and verifies that a device's behavior is consistent with known, authentic behavior over time.

Unlike traditional verification, it may be acceptable to consider a less rigid, but still meaningful definition. For example, a user that has a blood-pressure device may really only care if a blood-pressure monitor device is "hooked up" to the measurement app, and is operating in a way that is consistent with how a blood-pressure monitor should operate. Presumably, so long as as these properties hold, there is no immediate or obvious threat. If, however, a device connects as a blood-pressure monitor and then goes on to interact in a way that is inconsistent with typical interactions for this type of device, then there may be cause for concern. To this end, we examine various interpretations of verification with the end goal of realizing a more active form of security that can detect whether a device is no longer operating in a way that is consistent with expected behavior.

We see VIA's recurring verification of interaction patterns as a sort of second factor for authenticating the device. (The first factor being the presentation of typical identifiers and credentials, such as a `BD_ADDR` and session key derived from a long-term key; the second factor being an ongoing profile of its interactions validated against a previously learned model suitable for conducting verification.) As a result of this scheme, we introduce the notion of *recurring behavioral authentication for Bluetooth connections*, which can be integrated into a Bluetooth gateway device, such as a smartphone.

Contributions: We make the following contributions:

- ***A New Bluetooth Smart-Home and Smart-Health Dataset:*** We collected and present a new, first-of-its-kind dataset, which captures Bluetooth traces for app-device interactions between more than 20 smart-health and smart-home devices. (According to a recent survey [1] there are no such Bluetooth datasets; we found none despite an extensive search.) We share the dataset open-source [44].
- ***Extensions to Open-Source Bluetooth Software:*** We enhance open-source Bluetooth analysis software [46, 47] to improve the available tools for practical exploration of the Bluetooth protocol and Bluetooth-based apps.
- ***Adaptations of Traffic Modeling Techniques:*** We present a novel modeling technique (*hierarchical segmentation*) for characterizing and verifying authentic BLE app-device interactions.
- ***Implementation & Evaluation:*** We implemented the VIA design and technique and evaluate our approach against a test corpus of 20 smart-home and smart-health devices. Our results show that VIA can be used for verification using off-the-shelf machine-learning classifiers with an F1-score of 0.86 or better in most test cases.

2 BACKGROUND & RELATED WORK

Commodity smart-home and smart-health devices are generally *closed systems*; they generally cannot be extended with apps or services from third-party developers. As a result, typical solutions for protecting systems, such as installing third-party security software (e.g., antivirus software, security agents), are not applicable. In fact, in some cases – such as in FDA-approved medical devices – altering the software might invalidate the certification! This limitation – the inability to modify peripheral devices – has generated a lot of interest and work in *non-invasive* approaches to measuring the state of devices, and using the measurement results to determine whether devices are *authentic*. The detection of non-authentic devices through some form of measurements may be an indication of the presence of threats to apps, nearby devices, or the network itself. By detecting such threats, there is an opportunity to take action to secure these devices and networks. In the remainder of this section we review necessary background information and related work that helps to further motivate our methodology and contributions.

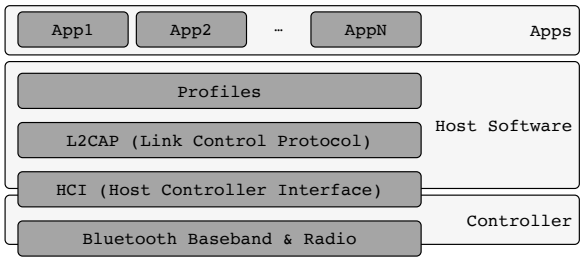

Figure 1: A simplified view of the Bluetooth stack.

2.1 Bluetooth

A typical deployment of Bluetooth consists of a Host and one or more Controllers (Figure 1). The Host Controller Interface (HCI) is an interface between the Host and Controllers. A Host is a logical entity made up of all the layers between Bluetooth's core profiles (i.e., Bluetooth apps and services) and the HCI. A Controller is a logical entity made up of all of the layers below the HCI, and enables the client to communicate with other Bluetooth devices.

Within the context of a shared, physical radio channel in Bluetooth, there is a complex layering of links and channels and associated control protocols that enables coordination amongst the devices as well as data to be transferred between devices [13, Volume 1, Part A]. Worthy of note are L2CAP channels, which provide a channel abstraction to apps and services, and are the primary means by which data is transported in Bluetooth Classic and BLE.

The hierarchy of links and channels within Bluetooth's architecture are arranged similarly in Bluetooth Classic and BLE. Therefore, we make references throughout this text to *Bluetooth*, with the understanding that it applies to both Bluetooth Classic and BLE.

2.2 Authentication

The overarching objective of our work (the verification of authentic interactions within WPANs) is largely motivated by past work in authentication [8, 12, 18, 22, 27, 31, 36–38, 40–42]. *Authentication* is a process to verify (i.e., establish the truth of) an attribute value claimed by or for a system entity [50]. In computer systems and networks, authentication is used to verify that a person (or another system) is in fact who or what it claims to be. This relates to our work in that VIA attempts to verify that apps and devices

interact in a way that is consistent with what they claim to be. Generally speaking, authentication can be achieved in one of two ways: *identification* or *verification*. Here, we focus on verification.

A system that performs *verification* either accepts or rejects an entity after examining some information about the entity's claimed identity. In this way, the task of verification is a one-to-one matching problem; i.e., the output of verification is binary: accept or reject. To perform verification: First, the system collects or obtains an identity and model for a particular entity, and stores it. Later, when an entity presents itself to the system and asserts its identity, it is the system's responsibility to verify that the presented information matches the model for that identity.

2.3 Intrusion Detection Systems (IDS)

Wireless traffic analysis systems and Intrusion Detection Systems have a long and rich history [2, 4, 6, 11, 12, 16, 23, 25, 26, 29, 33, 35, 39, 48, 55, 58]. Generally speaking, wireless traffic analysis systems and Intrusion Detection Systems (IDS) continually monitor computers or networks, collecting data (e.g., system calls, network communication), extracting quantifiable features from this data, and applying a variety of techniques to analyze the data in search of signs of anomalies or compromise.

With respect to IDSs, there are traditionally two broad categories of IDS with respect to *where* they are located: Host-based IDS (HIDS) and Network-based IDS (NIDS) [23]. HIDS are generally software located on the system being monitored, and typically monitor only that system. NIDS are often physically separate devices located somewhere "upstream" in the network of the system(s) being monitored. Furthermore, there are also two broad categories of IDS with respect to *how* intrusions are detected [53]: "systems relying on *misuse-detection* monitor activity with precise descriptions of known malicious behavior, while *anomaly-detection* systems have a notion of normal activity and flag deviations from that profile."

We opt for the NIDS approach and explore techniques commonly used to realize anomaly-based detectors. NIDS are advantageous in the context of our work because their presence is generally transparent to the systems being monitored, which means we could deploy a NIDS within a smartphone or other hub device to monitor interactions between apps and peripheral devices without needing to modify the apps or peripheral devices in any way. (This assumes that peripheral network traffic is routed through a network device – such as a smartphone or other hub device – where the NIDS is deployed, or that the NIDS has access to all network traffic, e.g., by sniffing an area of interest. This is reasonable in Bluetooth-based networks.) An anomaly-detection approach is advantageous in the context of our work because of the simplicity of peripheral devices, which generally serve a well-defined purpose and perform well-defined and often repetitive tasks (i.e., low variability).

2.3.1 Bluetooth & IDS. Few works have explored deploying an IDS within Bluetooth networks. The most closely-related work proposed a Mutli-Level Bluetooth IDS (ML-BIDS) [49] that envisions deploying a "whitelist" security mechanism coupled with an anomaly detection-based NIDS within a device such as a smartphone. Our work differs from ML-BIDS in a few significant ways: ML-BIDS is based on protocol state transition diagrams measured through the HCI protocol; VIA extracts features from packet headers and payloads, isolating anomalous observations in the application layer data and commands. There are also a number of differences in terms of system design: ML-BIDS is based on a "whitelisting" mechanism that relies on information specified by untrusted devices, and requires an administrator to actively manage security-critical information (e.g., a categorization of device criticality and data importance), and is dependent on a Master Whitelist Server (MWS). The design of VIA is more flexible and does not require active management by administrators, making it suitable for ad hoc networks common in smart-health and smart-home settings. Last, the evaluation of ML-BIDS is based on three devices (a single piconet made up of two devices plus an emulated attacker) and two old attacks (Bluesnarfing and power draining attacks), at least one of which (Bluesnarfing) was patched years ago. We evaluated VIA using a testbed of 20 modern smart-health and smart-home Bluetooth devices.

2.4 Malware Detection

We obtained invaluable insights from past work about side-channel analysis as an approach to detect malware [9, 10, 14, 17, 21, 30, 52, 54, 57]. Namely, in an IoT context where devices perform well-defined, repetitive tasks that should exhibit little variation from one run to another, measurements of network communications between apps and nearby devices may serve as a good proxy for computing activity (i.e., *behavior*). Using widely-accepted methods from machine learning, it may be possible to accurately model the well-defined (and often repetitive) network communications that occur in WPANs, such as Bluetooth networks. Assuming useful features can be extracted from the communications, it may be possible for a classifier to accurately detect a divergence from models for authentic communication, which may be indicative of a device whose firmware has changed, or of the presence of an inauthentic device masquerading as a legitimate device. Indeed, evidence suggests that a device infected with common types of malware will produce abnormal network activity (downloading files, creating new channels, opening ports, probing the mobile device, etc.) [17].

3 SYSTEM, NETWORK, & SECURITY MODEL

In this section we describe the context and scope of our work (Figure 2). For clarity, we highlight where VIA interposes on network communications, and we depict potentially untrusted components (red) within the hub and peripheral devices.

3.1 System & Network Model

Our work explores the viability of deploying anomaly-detection and intrusion-detection techniques within a hub device at the center of a WPAN. In this paper we consider WPANs that consist of multiple peripheral devices that connect with a central hub device. Hub devices in WPANs are often mobile devices (e.g., smartphones) but need not be mobile; rather, the role of this device is a system that runs end-user apps and serves as a gateway for other devices to access the Internet. In fact, this central gateway device might be a popular home "hub" device, such as Amazon Echo [3], Apple HomePod [5], or Google Home [24]; or a dedicated *health* hub device, such as the HealthGo Mini [20].

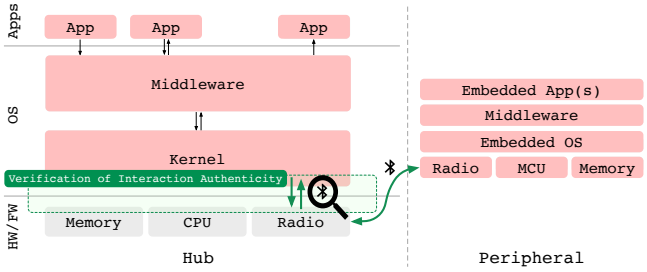

Figure 2: Overview of the VIA threat model and our solution.

IoT devices rely on the hub device to access the Internet (to upload data to services, or to interact with other IoT devices, for example) and to interact with the user (by presenting data to the owner/operator of the hub device). The hub devices rely on IoT devices as a means to collect data (health data from a worn device, for example) and to act on the environment (administer medication via an implanted insulin pump, for example).

3.2 Assumptions & Trust Model

We assume that any personal hub and IoT devices – when initially deployed – are deployed in a secure environment without any malware. Many IoT devices are deployed in, and generally exist in, an isolated, non-public environment (such as a home), which offers reasonable protection from physical threats; network-based threats may still be an issue, however (see below). Similar to past work (e.g., [32]), we assume VIA will be deployed within a trusted execution environment (TEE) such as SGX [28] or TrustZone [7], which ensures that VIA can reliably operate even if other system components (even the OS) are compromised.

We assume that IoT devices do not support the addition of third-party security mechanisms, such as antivirus, anti-malware, or any other security-related software agents. Furthermore, IoT devices have limited resources, such as energy, processing power, and memory. Hub devices are generally less constrained in terms of these resources, but any software deployed on a hub device should still be conscious of its impact on energy consumption, memory footprint, network usage, and so forth.

3.3 Threats & Adversary Model

The ultimate goal of the adversary is to compromise IoT and hub devices to steal sensitive data (e.g., personally identifiable information (PII), medical information, credentials), commandeer resources (e.g., device's computing resources), tamper with data (e.g., inject false data into apps), and to propagate itself (i.e., to leverage the compromised devices to carry out attacks against other devices). To clarify our threat model further, we describe two types of threats that our work specifically aims to address: *inbound threats* (peripheral-to-hub) and *outbound threats* (hub-to-peripheral).

3.3.1 Inbound Threats. Consider a malicious device that comes into close proximity (i.e., within Bluetooth's wireless radio range) of a target hub device. Here, a malicious device may be a *cloned device*, which is an attacker-controlled device that has cloned the identifiers of another device (e.g., MAC address, UUID(s), device name) with the objective of establishing a connection with the target hub device in an attempt to attack it. Or it may be a *compromised device*, which is a legitimate user-owned device that has been compromised by the attacker, and is now under the attacker's control. The objective of the attacker here may be to use this device to attack the target hub device or apps that run on the hub device.

3.3.2 Outbound Threats. Consider a malicious app present on the victim's hub device, which has the ability to access the Bluetooth interface. For example, on Android, this is as simple as an app having the `Bluetooth` and `Bluetooth_ADMIN` permission — two permissions that are claimed by almost all Bluetooth-capable apps [42]. Such a malicious app may, for example, use its access to the Bluetooth interface to attack nearby devices.

3.3.3 Scope & Limitations. As others have done (e.g., [17]), we focus our attention on *garden-variety* threats (e.g., generic malware that targets a vulnerability present in a large class of devices), which are a clear and present danger to hub and IoT devices. We acknowledge that an adversary with detailed knowledge of any defense mechanisms could in theory design an attack to specifically thwart or evade that defense mechanism.

It is important to note that VIA seeks to identify abnormal *behavior* by monitoring network communications. Because our work concentrates on behavioral features, our system will likely have a hard time detecting any difference between two devices that behave in similar ways, but run on physically different devices/hardware. For example, consider a model that accurately characterizes a heart-rate monitor used in remote patient care. Now consider an imposter heart-rate monitor device under the control of an attacker that "behaves correctly" (e.g., reports heart-rate values in a reasonable range) but is intentionally sending high heart-rate values (perhaps to convey a high resting heart rate to a medical record system, which may result in higher insurance costs due to high-risk health indicators). This imposter is in scope for our work, but we admit that we may not detect this sort of "attack" since the device is behaving within its normal profile. Thus, regardless of the underlying device/hardware – which may need to be verified by other means, such as platform attestation (e.g., [8]) – we only propose to identify authentic behavior vs. non-authentic behavior based on features related to communication between the devices. We are, however, currently unaware of realistic threats that operate in this way, and we suspect that attacks like the one described above are rare (or even nonexistent). We leave this issue to future work.

4 VERIFYING THE AUTHENTICITY OF INTERACTIONS IN WPANS

In this section we present *Verification of Interaction Authenticity (VIA)*, our approach to verifying trustworthy interactions (network communications) between devices within WPANs (Figure 3). We focus on the task of *verification* (Section 4.1) to introduce a new recurring authentication mechanism for ensuring that apps and devices continue to interact in a way that is consistent with prior observations (and is, with reasonable confidence, *authentic* and therefore *trustworthy*). To extract features from the network communications, we use a combination of n-grams (Section 4.2) and

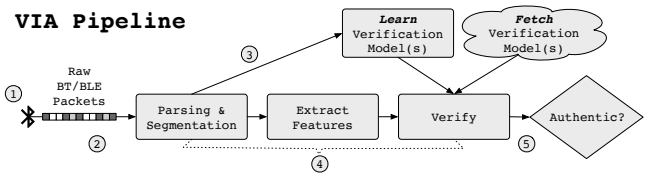

Figure 3: Overview of the VIA pipeline: (1) Establish connection between devices. (2) Observe peer device identity claim (e.g., BD_ADDR and LTK) at the time the connection is established. (3) Load the appropriate verification model. (4) Monitor/verify app-device interactions. (5) Depending on the verification result: if the interaction is deemed 'authentic', allow interaction to continue and repeat verification; otherwise, take action, such as *alert user* or *disconnect*.

our new methodology for model separation, *hierarchical segmentation* (Section 4.3). Deviation from authentic interactions will result in failed verification, enabling devices to take action and mitigate future threats and damage.

4.1 Verification

Authentication by way of verification (recall Section 2.2) requires two important steps: registration and the actual verification.

4.1.1 Registration. To accomplish verification, VIA first obtains many samples from a population of devices and learns a specific classifier for each device. With regards to Bluetooth, each device has an identity in the form of a BD_ADDR (though other forms of identity are possible, such as LTK), as well as a set of typical network interactions, which makes it possible to train a classifier that recognizes that device. While other strategies are possible, and other features may provide useful information (e.g., protocol state transitions [49]), in our work network interactions are currently represented with n-grams constructed from the contents of packet headers and payloads. Without loss of generality, we refer to some representation of typical network interactions as a *device profile*.

4.1.2 Verification Procedure. VIA monitors all network communications. When devices connect, VIA obtains a claimed-identity for the app-device interactions, and uses the identity to load the corresponding classifier obtained from registration (the *verification profile*). VIA then formulates a profile based on all newly-observed traffic (the *test profile*), and uses the classifier to conduct the verification that determines whether the test profile is sufficiently similar to the verification profile. In the end, VIA's verification procedure relies on the classifier to classify new observations into one of two classes: the *target* class or the *other* class. If the interactions are actually from a device that matches the identity of the target identity, the interactions should be classified to the *target* class, and VIA accepts the observations as being authentic. If, however, the observations are classified to the *other* class, VIA rejects the observations and and deems them to be inauthentic.

4.2 Network Traffic Modeling & Analysis

Our network traffic models are based on the contents of Bluetooth packet headers and/or payloads (Figure 7). This approach assumes that relevant packet contents are accessible to VIA – i.e., non-encrypted. Past work in Bluetooth (e.g., [45]) suggests that this is commonplace, and our results confirm this. Attacks commonly try to exploit vulnerabilities in services or apps by delivering maliciously crafted payloads; or, in the case of packet headers, attacks may try to exploit vulnerabilities in how the headers are parsed and interpreted. By modeling aspects of normal packets, it is possible to detect deviations in packet content that may indicate an attack.

To model network traffic in IP-based networks, past systems have successfully used n-grams (e.g., PAYL [55], PCkAD [4]). To describe data in terms of n-grams, we adopt the definition presented by Wressnegger et al. [56] and summarize it below. Each data object x first needs to be represented as a string of symbols from an alphabet, A, where A is often defined as bytes or tokens. For example, in modeling network packets, we simply consider a packet (or part of a packet, such as the packet headers or the packet payload) as a string of bytes. By moving a window of n symbols over the string of bytes in each packet x, we can then extract all substrings of length n. These substrings (n-grams) give rise to a map to a high-dimensional vector space, where each dimension is associated with the occurrences of one n-gram. Formally, this map ϕ can be constructed using the set S of all possible n-grams as,

$$\phi : x \to (\phi_s(x))_{s \in S} \quad \text{with} \quad \phi_s(x) = \text{occ}(s, x)$$

where the function occ(s, x) simply returns the frequencies, the probability, or a binary flag for the occurrences of the n-gram s in the data object x.

Past work has shown that even for small n (e.g., $n = 1$), n-grams can be an extremely effective in modeling traffic patterns in a manner that is efficient, accurate, and resilient to mimicry attacks [55]. (In a mimicry attack, the attacker (1) takes control of a network device, and (2) successfully delivers a payload while *mimicking* normal behavior. Thus, if an exploit sequence is contained in the normal profile, the attack will go undetected.) Attempts to improve upon simple 1-gram models have shown only slight improvements. For instance, past work has shown that slightly higher detection rates, and slightly lower false-positive rates can be achieved [4], but these improvements come at the expense of increased computational complexity, or the use of additional preprocessing that imposes domain knowledge. Furthermore, Angiulli et al. suggest that there is an inherent trade-off: greater values of n lead to higher false-positive rates, whereas 1-gram models have been shown to have lower detection rates (but not by much).

4.3 Hierarchical Segmentation

A single model trying to characterize *all* packets has been shown to lead to an ineffective, monolithic model [55]. Therefore, it is necessary to learn a variety of models that separate traffic into different groups, where similar types of traffic can be associated and compared. Past work analyzing IP traffic (e.g., [55]) learns separate models using a combination of destination port, packet payload length, and packet direction (inbound or outbound).[1] Thus, if there were 5 ports and 10 different payload lengths for each port,

[1] PAYL and related systems also describe solutions to common edge cases that can be problematic; e.g., they describe approaches to merge models when training data is sparse.

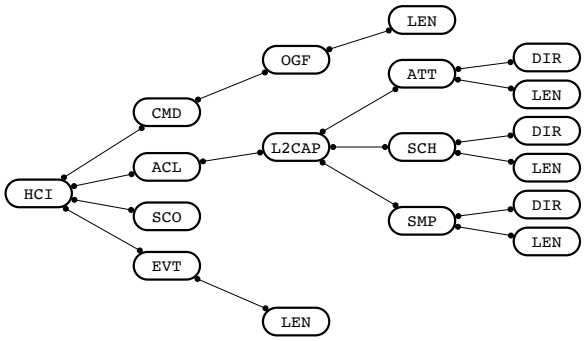

Figure 4: Hierarchical segmentation of the Bluetooth protocol stack for VIA models.

their approach would learn 50 separate models for inbound traffic and 50 separate models for outbound traffic.

This approach is problematic for Bluetooth-based WPANs. For instance, some of the conventional notions (e.g., ports) are not directly applicable. We observe, however, that model separation based on ports is effectively meant to separate models based on the underlying protocol and semantics of interactions within the context of a specific protocol. Therefore, to capture specific semantics of underlying protocols in Bluetooth, we introduce our approach to model separation based on *hierarchical segmentation* of the various Bluetooth protocol layers (Figure 4). Note that, as we traverse the hierarchy illustrated in Figure 4 from left to right, we move "up the stack" in Bluetooth. The root of the hierarchy corresponds to the HCI layer – the lowest layer of the Bluetooth protocol in which we can reliably capture Bluetooth network traffic.

The HCI protocol is made up of four different types of packets: command packets (CMD), which contain directives or requests from Host software; event packets (EVT), which contain responses to CMD requests or notifications of network events (e.g., connection requests); asynchronous data packets (ACL), which are used for higher layers in the Bluetooth protocol to exchange data; and synchronous data packets (SCO), which are primarily used for streaming data, such as audio. In our testbed of more than 20 smart-health and smart-home devices, not one of those devices ever exchanged SCO packets. Thus, VIA models currently only make use of CMD, EVT, and ACL packets.[2]

[2]Our approach to modeling should be applicable to SCO packets as well, but we have not had an opportunity to study SCO traffic to date. Although devices with microphones or speakers may use SCO, neither were included in any of our devices.

Each of the HCI packet types have substantially different characteristics in terms of directionality and packet length, as depicted in Figure 6. Generally speaking, HCI packets can flow in one of two directions. Packets that flow along the entry path into the hub are referred to as *ingress* packets (i.e., device-to-host or `d2h`), and packets that flow along the exit path from the hub are referred to as *egress* packets (i.e., host-to-device or `h2d`). CMD packets are unidirectional and flow exclusively from the Host to the Controller (`h2d`); the lengths of these packets are generally quite small, likely because the HCI protocol defines a a limited set of valid commands, most with few parameters. EVT packets flow exclusively from the Controller to the Host (`d2h`); the lengths of these packets are highly variable. ACL packets are bi-directional and can flow in either direction (`h2d` or `d2h`); these packets primarily transport application-layer data. There tends to be more variability in ingress ACL packets. These differences between CMD, EVT, and ACL packets are meaningful features for separation among VIA models.

ACL packets transport L2CAP packets, which provides the reliable transport layer for Bluetooth apps and their data (recall Section 2.1). Above the L2CAP layer, we identify three critical protocols that can be used for further model separation: the Attribute Protocol (ATT), the Signaling Protocol (SCH), and the Security Management Protocol (SMP). It is through these protocols that devices can perform authentication, establish connections and logical channels, and exchange user data.

To summarize, VIA uses several features to learn models: a combination of n-grams, packet type, packet length, and packet directionality. These features enable VIA to realize models that are highly effective in verifying whether network traffic is consistent with previously-learned models.

5 DATA COLLECTION

To evaluate our approach, we assembled a testbed (Figure 5) consisting of 9 different device types, and 20 devices in total (see Table 1 and Table 2). From this testbed, we produced a new dataset of more than 300 Bluetooth network traces. In this section we discuss the details of the testbed and how we produced our dataset.

5.1 A New Smart-Device Testbed

Our testbed (Figure 5) is currently concentrated around two broad categories of smart devices that are common in consumer WPANs: smart-health devices and smart-home devices. We carefully selected devices to ensure that our testbed was composed of a diverse range of devices in terms of their functions; yet, we also wanted to evaluate potential limitations of our approach in modeling, and differentiating among, similar devices.

Throughout the remainder of the paper, we describe devices by their ***type***, which refers to a device's functionality and purpose; ***make***, which refers to the manufacturer of the device; and ***model***, which refers to an identifier, such as a name or number, that is used to distinguish among devices made by the same manufacturer. A summary of the devices in our testbed is presented in Table 1 along with a summary of the relevant apps in Table 2.

Table 1: A Bluetooth-enabled smart-device testbed.

Identifier	Device Model
Smart Health	
BP Monitor iHealth [wrist] (1)	iHealth View Bluetooth Wrist Blood Pressure Monitor
BP Monitor iHealth [upperarm] (1)	iHealth Feel Bluetooth Upper Arm Blood Pressure Monitor
BP Monitor Omron [wrist] (1)	OMRON 10 Series Wireless Wrist Blood Pressure Monitor
BP Monitor Omron [upperarm] (1)	OMRON Evolv Wireless Upper Arm Blood Pressure Monitor
BP Monitor Choice [upperarm] (1)	Choice Wireless Blood Pressure Monitor, Upper Arm
Glucosemonitor iHealth [na] (1)	iHealth Wireless Smart Blood Sugar Test Kit
Glucosemonitor Choice [na] (1)	Choice Wireless Blood Glucose Monitor
HR Monitor PolarH7 [chest] (1)	Polar H7 Wearable Heart Rate Monitor (Chest)
HR Monitor PolarH7 [chest] (2)	Polar H7 Wearable Heart Rate Monitor (Chest)
HR Monitor Zephyr [chest] (1)	Zephyr Wearable Heart Rate Monitor (Chest)
Pulse Oximeter iHealth [finger] (1)	iHealth Air Wireless Fingertip Pulse Oximeter
Scale Gurus [floor] (1)	Bluetooth Smart Body Fat Scale by Weight Gurus
Scale Renpho [floor] (1)	RENPHO Smart Bluetooth Body Fat Scale
TENS Unit Omron [na] (1)	OMRON Avail Dual Channel TENS unit
Thermometer Kinsa [ear] (1)	KINSA Smart Ear (in-ear smart thermometer)
Thermometer Kinsa [oral] (1)	KINSA QuickCare (oral smart thermometer)
Smart Home	
Env Sensor Inkbird [na] (1)	Inkbird combo mini Bluetooth (temp/hum) sensor
Env Sensor Inkbird [na] (2)	Inkbird combo mini Bluetooth (temp/hum) sensor
Smart Lock August [door] (1)	August Smart Lock Pro + Connect (3^{rd} Gen.)
Smart Lock Schlage [door] (1)	Schlage Sense Smart Deadbolt

Format: Device & Manufacturer [Location] (Device ID)

Table 2: List of smart-device companion apps.

App	Corresponding Device(s)
Smart Health	
RENPHO	RENPHO scale
Weight Gurus	Weight Gurus scale
iHealth MyVitals	iHealth blood-pressure monitors, pulse oximeter
OMRON Connect	OMRON blood-pressure monitors
Choice Blood Pressure	Choice blood-pressure monitor
Polar Beat	Polar and Zephyr heart-rate monitors
OMRON TENS	OMRON TENS unit
iHealth Gluco-Smart	iHealth blood-glucose meter
AgaMatrix Diabetes Manager	Choice blood-glucose meter
Kinsa	Kinsa oral and ear thermometers
Smart Home	
Schlage Home	Schlage smart deadbolt
August Home	August smart lock
Engbird	Environment sensors

5.2 Capturing HCI Traces

Our dataset consists of a collection of Bluetooth HCI network traces that capture interactions between 20 distinct devices with 13 different smartphone apps. Here, a ***trace*** refers to a packet capture consisting of all packets that are observed between the time that a hub and peripheral device establish and terminate a connection.

The above-mentioned smartphone apps were installed on a Nexus 5 smartphone running Android 6.0.1 ("Marshmallow"), API level 23, kernel version 3.4.0. Along with executing the apps, the smartphone also served as our primary device for data collection. To capture HCI traces, we enabled the Bluetooth *HCI snoop log* developer option. (This feature is a common developer option introduced in Android 4.4. It is interesting to note that using this feature does not even require rooting the phone.) The HCI snoop log captures all Bluetooth HCI packets to a binary-encoded file, which it writes to an SD card; the log format resembles the Snoop Version 2 Packet Capture File Format described in RFC 1761 [15].

Each trace captured interactions between one app-device pair. Specifically, each trace captured *all* communications observed at the HCI layer (and therefore all protocol layers above the HCI layer). For each app-device pair we collected at least 10 traces, each of which included 3-10 minutes of network activity.

5.3 Emulating Normal App-Device Interactions

We gathered HCI traces by manually using the apps and devices in our testbed to emulate a wide variety of normal app-device interactions. The actions we performed consisted of: navigating the "official" smartphone app[3] and exercising features that trigger network communication with a corresponding device, as well as acting upon the devices in such a way that triggers communication with its corresponding smartphone app. This usage generated traffic that was observable at the HCI layer, including traffic related to connection/disconnection events and application-layer data exchanges via the L2CAP and ATT protocols. In general, by concentrating primarily on the HCI→ACL branch of hierarchical segmentation (Figure 4), and specifically the L2CAP layer and above, our solution primarily captures typical device behavior mixed with user-dependent data.

It was not our intention to discover and exercise *every* functional feature (and thus every BLE service or characteristic) of a particular app/device. Rather, it was our intention to observe *typical* features and interactions between devices and their official app, which could be used to construct normality models suitable for performing verification in future interactions. Our current stance is that any network activity (authentic or inauthentic, benign or harmful) that deviates significantly from our normality models should fail verification, which provides an opportunity for further investigation or some other response. Enhancing the dataset with data from more apps/devices, and more exhaustive usage, is an area of future work.

5.4 Data Pre-processing

Prior to analysis we applied pre-processing to each raw trace file. Data pre-processing is necessary to generate an intermediate file with per-packet features and labels. More specifically, after collecting each HCI trace (an *HCI snoop file*), we moved the raw file from the smartphone to a local VM (running Linux, kernel version 4.14.) using the Android Debugger command line tool (`adb`). During data collection, we reset the HCI snoop file on the smartphone between each trace so that each HCI snoop file contained only packets belonging to interactions between a particular app-device combination; we refer to this as an *app-device session*.

To parse HCI traces, we extended two open-source projects: `bluepy` [46] and `btsnoop` [47]. Our extensions extend the parsing of the HCI protocol and other protocols that the HCI protocol encapsulates; namely, we extract features for each packet within the HCI traces, such as packet types, lengths, endpoint identifiers, protocol semantics, and segmented headers and payloads belonging to higher-level Bluetooth protocols (e.g., ATT, SCH, SMP); and, because each trace captured a single app-device session, we labeled each packet according to the device it was sent to/from. These features and device labels were written to CSV-formatted files for subsequent analysis.

6 EVALUATION & RESULTS

This section describes our experiments that use the traces collected from our smart-device testbed. The primary thrust of our experiments is to show that our approach can produce models capable of differentiating between sufficiently dissimilar devices, which is necessary to conduct verification. To this end, we evaluated VIA in terms of its ability to perform verification tasks at different *granularities* of a device's identify. Specifically, we considered three granularities: device type, device type *and* make, and specific device instance.[4] We also conducted experiments to examine various special cases where verification is less successful. For example, VIA's verification performs worse in cases where nearly identical devices (i.e., devices of the same type, make, and model) exist in both the *target* and *other* classes.

[3] With the exception of the heart-rate monitors, each device had a single "official" smartphone app to which it would connect. Thus, we assume each device has a single official application and intends to communicate with that application only. An application, however, may communicate with one or more devices. For example, iHealth Labs developed a single application for its users (iHealth MyVitals) that interacts with *all* iHealth devices; in this case there is a 1:many app-device relational mapping.

[4] We do not evaluate the granularity of device type, make, and model (*device-type-make-model*) here. While it may seem a natural next step, our testbed did not contain enough examples to provide a meaningful evaluation at this class granularity.

6.1 Experimental Procedure

To conduct our experiments we loaded the pre-processed files from our dataset into $m \times n$ matrices, where the rows are the observations (packets) in the trace, and the columns are the features (e.g., packet types, lengths, raw packet bytes) parsed from the raw trace files. We then segmented each packet according to the hierarchical segmentation methodology described in Section 4.3, and computed n-grams over the packet bytes in the respective segments.

One practical issue here is that packets exchanged in Bluetooth are usually quite small. For example, in Bluetooth version 4.0 and 4.1, the maximum ATT payload is 20 bytes. Thus, we have found that learning models and conducting verification works best using a sequence of packets. In our work we chose to use all of the available bytes/packets in a network trace[5] to learn and evaluate our models, which equates to a few minutes of network activity per trace.

Since there are more than two classes for each granularity we considered, we followed a *one-vs-all* strategy to decompose the problem into a binary classification problem. Namely, we repeatedly fixed a single class to be the *target* class and combine all other classes into a single *other* class.[6] We then evaluated the verification results produced by VIA using the following definitions. A successful verification is defined as a classification of observations belonging to the *target* class into the *target* class (true positive), or a classification of observations belonging to the *other* class into the *other* class (true negative). An unsuccessful verification is defined as a classification of observations belonging to the *target* class into the *other* class (false negative), or a classification of observations belonging to the *other* class into the *target* class (false positive).

To learn and evaluate models, we experimented with various machine learning classifiers and different "branches" within hierarchical segmentation, to classify a sequence of packets into one of two classes. For brevity we report results using only the Random Forest classifier, which we found to be a suitable classifier after comparing its performance to a collection of other classifiers. In the remainder of this section, we present our experimental results, which were trained and validated with n-grams computed from the HCI→ACL→L2CAP→ $\cdots$ branch of hierarchical segmentation.

To evaluate the performance of the various classifiers examined in this work, we conducted a stratified 10-fold cross validation that repeatedly split our dataset into training and testing subsets. The classifier was trained with n-grams and class labels from a training set. We then used the trained models to output a class label for each sample in a corresponding test set (i.e., the n-grams computed from a new trace) and compared the output label to the actual label.

[5]The decision to use all of the bytes in a network trace is appropriate here because of how we constructed the traces (recall Section 5.2). In short, each trace contains bytes from a short period of time (3-10 minutes) in which an app-device pair connect, exchange data, and disconnect. In future work, we will perform an in-depth analysis to determine an optimal duration that should be used for building a model.

[6]To date, we have not yet evaluated how VIA performs when presented with an unknown device (i.e., a device not seen in training). In VIA's current design, any observation from an unknown device will be classified to the class it most closely resembles; ideally this class would be the *other* class, but since the unknown observation is represented in neither the *target* class nor the *other* class, VIA's behavior in this scenario is unknown. One solution to this issue is to implement a confidence threshold before allowing an observation to be classified into the *target* class. If the new observation does not exceed the confidence threshold, VIA would classify the observation into the *other* class, meaning that the observation does not look sufficiently similar to the *target* class for verification. We plan to add and evaluate this enhancement to VIA's verification capabilities in future work.

6.2 Verification Experiments & Results

First, we present specific experimental details and results for conducting verification at varying granularities.

6.2.1 Verification Task #1: Classification by Device Type. In this experiment we investigated whether VIA can verify that a new sample belongs to a specific device type (*device-type*). To evaluate verification by device type, we re-labeled each trace according to a *one-vs-all* strategy (Section 6.1) and conducted a stratified 10-fold cross-validation using models learned from the re-labeled data. We repeated this process 9 times to produce results where each device type was fixed as the *target* class. The results for each of the 9 *device-type* classes are summarized in Table 3. In the *device-type* verification tasks, VIA achieved an F1-score of 0.90 or better in all *device-type* classes.

Table 3: The precision, recall, and F1-score for one-vs-all verification by device type (*device-type*).

Device Type	Precision	Recall	F1-Score
BP Monitor	1.0	0.96	0.98
Env Sensor	1.0	0.81	0.90
Glucosemonitor	1.0	0.96	0.98
HR Monitor	1.0	0.91	0.95
Pulse Oximeter	1.0	1.00	1.00
Scale	1.0	0.90	0.95
Smart Lock	1.0	1.00	1.00
TENS Unit	1.0	0.92	0.96
Thermometer	1.0	0.97	0.98

6.2.2 Verification Task #2: Classification by Device Type and Make. In this experiment we investigated whether VIA can verify that a new sample belongs to a specific device type and make (*device-type-make*). To evaluate verification by device type and make, we re-labeled each trace according to a *one-vs-all* strategy (Section 6.1) and conducted a stratified 10-fold cross-validation using models learned from the re-labeled data. We repeated this process 15 times to produce results where each device type and make was fixed as the *target* class. The results for each of the 15 *device-type-make* classes are summarized in Table 4. In the *device-type-make* verification tasks, VIA achieved an F1-score of 0.92 or better in nearly all *device-type-make* classes. Indeed, the only cases where VIA performed worse were tasks where VIA had to perform verification when similar devices (e.g., heart-rate monitors) were in both the *target* class and the *other* class. We discuss this observation in Section 6.2.4.

6.2.3 Verification Task #3: Classification by Device Instance. In this experiment we investigated whether VIA can verify that a new sample belongs to a specific device (*device-instance*). To evaluate verification by device instance, we re-labeled each trace according to a *one-vs-all* strategy (Section 6.1) and conducted a stratified 10-fold cross-validation using models learned from the re-labeled data. We repeated this process 20 times to produce results where each device instance is fixed as the *target* class. The results for each of the 20 *device-instance* classes are summarized in Table 5. In the *device-instance* verification tasks, VIA achieved an F1-score of 0.86 or better in nearly all *device-instance* classes. Again, the only cases where VIA

Table 4: The precision, recall, and F1-score for one-vs-all verification by device type and make (*device-type-make*).

Device Group (Type & Make)	Precision	Recall	F1-Score
BP Monitor Choice	1.0	1.00	1.00
BP Monitor iHealth	1.0	0.90	0.95
BP Monitor Omron	1.0	1.00	1.00
Env Sensor Inkbird	1.0	0.94	0.97
Glucosemonitor Choice	1.0	1.00	1.00
Glucosemonitor iHealth	1.0	0.91	0.95
HR Monitor PolarH7	0.9	0.86	0.88
HR Monitor Zephyr	1.0	0.75	0.86
Pulse Oximeter iHealth	1.0	1.00	1.00
Scale Gurus	1.0	0.90	0.95
Scale Renpho	1.0	0.90	0.95
Smart Lock August	0.9	0.95	0.93
Smart Lock Schlage	1.0	0.86	0.92
TENS Unit Omron	1.0	0.92	0.96
Thermometer Kinsa	1.0	0.97	0.98

performed worse were tasks where VIA had to perform verification when nearly identical devices (e.g., environment sensors) were in both the *target* class *and* the *other* class.

Table 5: The precision, recall, and F1-score for one-vs-all verification by device instance (*device-instance*).

Device Instance	Precision	Recall	F1-Score
BP Monitor Choice [upperarm] (1)	1.00	1.00	1.00
BP Monitor iHealth [upperarm] (1)	1.00	0.92	0.96
BP Monitor iHealth [wrist] (1)	1.00	0.94	0.97
BP Monitor Omron [upperarm] (1)	1.00	1.00	1.00
BP Monitor Omron [wrist] (1)	1.00	1.00	1.00
Env Sensor Inkbird [na] (1)	0.50	0.43	0.46
Env Sensor Inkbird [na] (2)	0.50	0.22	0.31
Glucosemonitor Choice [na] (1)	1.00	1.00	1.00
Glucosemonitor iHealth [na] (1)	1.00	0.91	0.95
HR Monitor PolarH7 [chest] (1)	1.00	0.45	0.62
HR Monitor PolarH7 [chest] (2)	1.00	0.18	0.31
HR Monitor Zephyr [chest] (1)	1.00	0.75	0.86
Pulse Oximeter iHealth [finger] (1)	1.00	1.00	1.00
Scale Gurus [floor] (1)	1.00	0.90	0.95
Scale Renpho [floor] (1)	1.00	0.80	0.89
Smart Lock August [door] (1)	0.91	1.00	0.95
Smart Lock Schlage [door] (1)	1.00	0.86	0.92
TENS Unit Omron [na] (1)	1.00	0.92	0.96
Thermometer Kinsa [ear] (1)	1.00	1.00	1.00
Thermometer Kinsa [oral] (1)	1.00	0.94	0.97

6.2.4 Discussion: Verification at Various Granularities. In our evaluation of VIA's ability to perform verification tasks, it is important to understand that the F1-score is lower in the *device-type-make* evaluation and the *device-instance* evaluation because similar (sometimes *nearly identical*) devices are in both the *target* and *other* classes. For instance, when attempting to verify a Zephyr heart-rate monitor in the *device-type-make* evaluation, there are heart-rate monitors in both the *target* class (the Zephyr heart-rate monitor) and *other* class (the Polar7 heart-rate monitors). As another example, when attempting to verify an Inkbird environment sensor in the *device-instance* evaluation, there is one instance of an environment sensor in both the *target* class and *other* class. These particular observations are not surprising. The similar devices – and indeed the nearly identical devices – operate in similar ways, and in some cases even run the same software, on independent (but otherwise identical) hardware. It is therefore unsurprising to observe that VIA confused these devices when performing certain verification tasks.

One could conclude that VIA's verification is best done at the granularity of *device-type*; however, we urge the reader to consider the consequences of this conclusion. For example, the consequence of verification at the granularity of *device-type* is that VIA may not take into account important distinctive features that arise from devices with different make, model, or instance. Furthermore, we would be remiss to not point out that VIA's perceived success in performing verification may be exaggerated by the fact that our testbed contained few devices that shared the same type, make *and* model – the cases where VIA is most likely to be confounded.

In light of these observations, we conclude that further analysis is needed to better understand the strengths and limitations of our approach, specifically with respect to highly-similar devices.

6.3 Similarity Experiments & Results

In the experiments we discuss next, we examine cases where devices were highly similar (e.g., same device type, make, model).

6.3.1 Same Type, Different Manufacturer. In this experiment we examine VIA's ability to distinguish between devices that are functionally similar (same type), but made by different manufacturers. We focused on five different blood-pressure monitors from three different manufacturers (Omron, iHealth, and Choice) and conducted a comparative analysis among these devices only. We trained and evaluated models for each of the five instances (classes). The results are summarized in Table 6.

The results show that our approach yields coherent differentiation between the models learned for devices from different manufacturers. It is also evident, however, that there was confusion between similar devices made by the same manufacturer. For instance, in this experiment, we used two blood-pressure monitors from iHealth; one takes measurements from the wrist while the other takes measurements from the upper arm. From the observed network traffic between the iHealth app and the iHealth blood-pressure monitors, these devices appeared to be nearly identical. A similar discussion applies for the Omron wrist and upper arm devices. This result suggests that VIA will likely have difficulty distinguishing between devices of the same type, made by the same manufacturer. This outcome is not unexpected, however. It makes sense that device makers would use similar hardware and software (perhaps even *the same* underlying hardware and software) to build their various devices. Thus, our takeaway is that VIA is in fact capable of verifying that traces belonging to similar devices from the same manufacturer are consistent with a previously learned model.

We conclude that, even though the devices themselves are similar in terms of the function they provide to the end-user (e.g.,

blood-pressure measurements), VIA can reliably distinguish between functionally-similar devices that are made by *different* manufacturers. This strongly suggests that it would be difficult for adversaries to successfully masquerade themselves as authentic devices *at the same time* that they carry out any other nefarious actions (since even highly similar devices are rarely misclassified).

Table 6: The precision, recall, and F1-score for classification of blood-pressure monitors.

Blood-Pressure Monitor Instance	Precision	Recall	F1-Score
BP Monitor Choice [upperarm] (1)	1.00	1.00	1.00
BP Monitor iHealth [upperarm] (1)	1.00	0.92	0.96
BP Monitor iHealth [wrist] (1)	0.94	1.00	0.97
BP Monitor Omron [upperarm] (1)	1.00	1.00	1.00
BP Monitor Omron [wrist] (1)	1.00	1.00	1.00

6.3.2 Same Type, Same Manufacturer. In this experiment, we used two Inkbird environment sensors to investigate VIA's ability to distinguish among distinct instances of the same device (i.e., devices that are identical in their hardware, firmware, and software). We trained and evaluated models based on each of the two instances (classes). The results are summarized in Table 7.

The results here reveal that VIA struggled to distinguish between these highly similar devices. This insight, however, is not surprising, and in fact, it highlights a benefit of our approach; namely, that a trusted manufacturer could provide a device profile that can be used reliably for their devices (i.e., it may not be necessary to produce different device profiles for different instances of the same device.) These devices are identical in terms of their software and hardware; it therefore makes sense that these devices would exhibit similar communication patterns and packet contents. While this does point to a potential limitation of VIA's ability to distinguish between distinct devices, we argue that this result does not undermine the value of VIA. Because our objective is to verify the authenticity of interactions between apps and devices, confusion between profiles for an app and two devices that appear to function and communicate in nearly identical ways does not pose an immediately obvious threat to a WPAN.

Table 7: The precision, recall, and F1-score for classification of environment sensors.

Environment Sensor Instance	Precision	Recall	F1-Score
Env Sensor Inkbird [na] (1)	0.60	0.43	0.50
Env Sensor Inkbird [na] (2)	0.64	0.78	0.70

7 DISCUSSION

In this section we discuss some of the challenges and opportunities surrounding real-world deployments of VIA.

7.1 Bootstrapping Initial Verification Models

In practice, a deployment of VIA must address how initial verification models are obtained. We envision a combination of two likely approaches (recall Figure 3). One approach is to learn verification models on-the-fly. For example, when a new, previously-unseen peripheral device connects with a hub running VIA, some number of initial interactions can be used to learn the models for authentic interactions. After this initial phase, the learned models can be used for performing verification of subsequent interactions.

Another approach is to obtain verification models from a trusted source, such as a device manufacturer or software/firmware vendor. These models could be fetched based on device identifiers that are exchanged when two devices initiate a connection establishment procedure. Similar to on-the-fly learning, after the appropriate model(s) have been retrieved, they can be used for performing verification of subsequent interactions. We note that this approach is gaining traction today: for example, MUD [34] is a proposed IETF standard for formally specifying the expected network behavior of an IoT device. MUD exploits a similar observation that we do: IoT devices (generally) perform a limited set of functions, and therefore have a recognizable communication pattern that can be represented by a model (referred to as a *MUD profile* in the context of MUD).

7.2 Response Strategies

Our work thus far focuses on techniques to verify authentic interactions, but an open challenge is how to best respond when verification fails. Likely strategies are: "flag" the device and increase the verification requirements to make them more strict going forward; discard data that is not from a verified source; alert the user (e.g., mobile notification, SMS, email, auto-generated report to technical auditor); disconnect the app/device connection; and/or temporarily disable the network interface altogether. To mitigate usability issues that may arise from false positives (i.e., failed verification that is not due to malicious activity), VIA's responsive actions could initially be more lenient and become more aggressive if there is continued deviation from a verification model over time.

8 CONCLUSION

In this paper we present *Verification of Interaction Authenticity (VIA)*, a new *device-to-device* recurring authentication scheme based on evaluating characteristics of the communications (*interactions*) between devices within wireless networks. We specifically evaluate our approach in the context of Bluetooth networks. We present a new Bluetooth dataset consisting of more than 300 Bluetooth network traces, along with an experimental implementation and evaluation of VIA. We found that devices can be correctly verified at a variety of granularities, achieving an F1-score of 0.90, 0.92, 0.86 or better in most cases. We also evaluate potential limitations of our approach, but conclude that many edge cases that we evaluated do not actually impact the utility of VIA.

ACKNOWLEDGEMENTS

This research results from a research program at the Institute for Security, Technology, and Society at Dartmouth College, supported by the National Science Foundation under award number CNS-1329686 and CNS-1955805. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the sponsors.

REFERENCES

[1] M. Ahmed, A. Naser Mahmood, and J. Hu. A survey of network anomaly detection techniques. *Journal of Network and Computer Applications*, v.60 pages 19–31, 2016. DOI 10.1016/j.jnca.2015.11.016.

[2] S. Al-Riyami, F. Coenen, and A. Lisitsa. A Re-evaluation of Intrusion Detection Accuracy: Alternative Evaluation Strategy. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 2195–2197. ACM Press, 2018. DOI 10.1145/3243734.3278490.

[3] Amazon. Amazon Echo, 2018. Online at https://www.amazon.com/echo/.

[4] F. Angiulli, L. Argento, and A. Furfaro. Exploiting N-gram location for intrusion detection. *Proceedings of the IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, v.2016-Janua pages 1093–1098, 2015. DOI 10.1109/ICTAI.2015.155.

[5] Apple. Apple HomePod, 2018. Online at https://www.apple.com/homepod/.

[6] H. Arai, K. Emura, and T. Hayashi. A Framework of Privacy Preserving Anomaly Detection. In *Proceedings of the Workshop on Privacy in the Electronic Society (WPES)*, pages 111–122. ACM Press, 2017. DOI 10.1145/3139550.3139551.

[7] Arm. Arm TrustZone, 2018. Online at https://www.arm.com/products/silicon-ip-security.

[8] N. Asokan, F. Brasser, A. Ibrahim, A.-R. Sadeghi, M. Schunter, G. Tsudik, and C. Wachsmann. SEDA: Scalable Embedded Device Attestation. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 964–975. ACM Press, 2015. DOI 10.1145/2810103.2813670.

[9] A. J. Aviv, B. Sapp, M. Blaze, and J. M. Smith. Practicality of Accelerometer Side Channels on Smartphones. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC)*, pages 41–50. ACM, 2012. DOI 10.1145/2420950.2420957.

[10] D. Balzarotti, M. Cova, and G. Vigna. ClearShot: Eavesdropping on Keyboard Input from Video. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, pages 170–183, May 2008. DOI 10.1109/SP.2008.28.

[11] D. Bekerman, B. Shapira, L. Rokach, and A. Bar. Unknown Malware Detection Using Network Traffic Classification. In *Proceedings of the IEEE Conference on Communications and Network Security (CNS)*, pages 134–142, 2015. Online at https://cyber.bgu.ac.il/wp-content/uploads/2017/10/07346821.pdf.

[12] B. Bezawada, M. Bachani, J. Peterson, H. Shirazi, I. Ray, and I. Ray. Behavioral Fingerprinting of IoT Devices. In *Proceedings of the Workshop on Attacks and Solutions in Hardware Security (ASHES)*, pages 41–50. ACM Press, 2018. DOI 10.1145/3266444.3266452.

[13] Bluetooth. Bluetooth Specifications. Online at https://www.bluetooth.com/specifications.

[14] E. Brier, C. Clavier, and F. Olivier. Correlation Power Analysis with a Leakage Model. In M. Joye and J.-J. Quisquater, editors, *Cryptographic Hardware and Embedded Systems (CHES)*, volume 3156 of *Lecture Notes in Computer Science*, pages 16–29. Springer Berlin Heidelberg, 2004. DOI 10.1007/978-3-540-28632-5_2.

[15] B. Callaghan and R. Gilligan. Snoop Version 2 Packet Capture File Format, 1995. Online at https://tools.ietf.org/html/rfc1761.

[16] J. Camacho, P. García-Teodoro, and G. Maciá-Fernández. Traffic Monitoring and Diagnosis with Multivariate Statistical Network Monitoring: A Case Study. In *IEEE Security and Privacy Workshops (SPW)*, pages 241–246. IEEE, May 2017. DOI 10.1109/SPW.2017.11.

[17] S. S. Clark, B. Ransford, A. Rahmati, S. Guineau, J. Sorber, W. Xu, K. Fu, A. Rahmati, M. Salajegheh, and D. Holcomb. WattsUpDoc: Power Side Channels to Nonintrusively Discover Untargeted Malware on Embedded Medical Devices. In *USENIX Workshop on Health Information Technologies*, pages 221–236, 2013.

[18] C. Cornelius. *Usable Security for Wireless Body-Area Networks*. PhD thesis, Dartmouth College, 2013.

[19] C. Crane. 20 Surprising IoT Statistics You Don't Already Know, 2019. Online at https://securityboulevard.com/2019/09/20-surprising-iot-statistics-you-dont-already-know/.

[20] EDevice. HealthGo Mini, 2018. Online at https://www.edevice.com/products/healthgo-mini.

[21] D. Genkin, L. Pachmanov, I. Pipman, A. Shamir, and E. Tromer. Physical Key Extraction Attacks on PCs. *Communications of the ACM*, 59(6) pages 70–79, May 2016. DOI 10.1145/2851486.

[22] S. Gisdakis, T. Giannetsos, and P. Papadimitratos. SHIELD: A Data Verification Framework for Participatory Sensing Systems. In *Proceedings of the Conference on Wireless Security (WiSec)*, 2015. DOI 10.1145/2766498.2766503.

[23] T. R. Glass-Vanderlan, M. D. Iannacone, M. S. Vincent, Qian, Chen, and R. A. Bridges. A Survey of Intrusion Detection Systems Leveraging Host Data. *Computing Research Repository (CoRR)*, v.abs/1805.0 pages 1–40, 2018. Online at https://arxiv.org/abs/1805.06070v2.

[24] Google. Google Home, 2018. Online at https://madeby.google.com/home/.

[25] G. Gu, R. Perdisci, J. Zhang, and W. Lee. BotMiner: Clustering Analysis of Network Traffic for Protocol-and Structure-Independent Botnet Detection. In *USENIX Security*, 2008. Online at http://faculty.cs.tamu.edu/guofei/paper/Gu_Security08_BotMiner.pdf.

[26] H. Hindy, D. Brosset, E. Bayne, A. Seeam, C. Tachtatzis, R. Atkinson, and X. Bellekens. A Taxonomy and Survey of Intrusion Detection System Design Techniques, Network Threats and Datasets. *Computing Research Repository (CoRR)*, pages 1–35, 2018. Online at http://arxiv.org/abs/1806.03517.

[27] A. Ibrahim. Securing Embedded Networks through Secure Collective Attestation. In *MobiSys PhD Forum*, pages 1–2, 2018.

[28] Intel. Intel Software Guard Extensions (SGX), 2018. Online at https://software.intel.com/sgx/.

[29] R. A. Kemmerer and G. Vigna. Intrusion Detection: A Brief History and Overview. *Computer: Security & Privacy*, 35(4) pages 27–30, 2002. DOI 10.1109/MC.2002.1012428.

[30] P. C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *Advances in Cryptology — CRYPTO*, volume 1109 of *Lecture Notes in Computer Science*, chapter 9, pages 104–113. Springer Berlin Heidelberg, Jul. 1996. DOI 10.1007/3-540-68697-5_9.

[31] K. Kostiainen, E. Reshetova, J.-E. Ekberg, and N. Asokan. Old, New, Borrowed, Blue – A Perspective on the Evolution of Mobile Platform Security Architectures. In *Proceedings of the ACM Conference on Data and Application Security and Privacy (CODASPY)*, pages 13–24. ACM Press, 2011. DOI 10.1145/1943513.1943517.

[32] D. Kuvaiskii, S. Chakrabarti, and M. Vij. Snort Intrusion Detection System with Intel Software Guard Extension (Intel SGX). In *arXiv preprint arXiv:1802.00508*, 2018. Online at http://arxiv.org/abs/1802.00508.

[33] A. Lakhina, M. Crovella, and C. Diot. Diagnosing Network-Wide Traffic Anomalies. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pages 219–230, 2004. DOI 10.1145/1030194.1015492.

[34] E. Lear, R. Droms, and D. Romascanu. Manufacturer Usage Description Specification, 2019. Online at https://tools.ietf.org/html/rfc8520.

[35] M. V. Mahoney. Network Traffic Anomaly Detection Based on Packet Bytes. In *Proceedings of the ACM Symposium on Applied Computing (SAC)*, pages 1–5, 2003. DOI 10.1145/952589.952601.

[36] A. L. Maia Neto, A. L. F. Souza, I. Cunha, M. Nogueira, I. O. Nunes, L. Cotta, N. Gentille, A. A. F. Loureiro, D. F. Aranha, H. K. Patil, and L. B. Oliveira. AoT: Authentication and Access Control for the Entire IoT Device Life-Cycle. In *Proceedings of the ACM Conference on Embedded Network Sensor Systems (SenSys)*, pages 1–15. ACM, 2016. DOI 10.1145/2994551.2994555.

[37] S. Mare. *Seamless Authentication For Ubiquitous Devices*. PhD thesis, Dartmouth College, 2016. Online at http://www.cs.dartmouth.edu/reports/abstracts/TR2016-793/.

[38] S. Mare, R. Rawassizadeh, R. Peterson, and D. Kotz. Continuous Smartphone Authentication using Wristbands. In *Proceedings of the Workshop on Usable Security and Privacy (USEC)*, 2019. DOI 10.14722/usec.2019.23013.

[39] Y. Meidan, M. Bohadana, A. Shabtai, J. D. Guarnizo, M. Ochoa, N. O. Tippenhauer, and Y. Elovici. ProfilIoT: A Machine Learning Approach for IoT Device Identification Based on Network Traffic Analysis. In *Proceedings of the Symposium on Applied Computing (SAC)*, pages 506–509. ACM Press, 2017. DOI 10.1145/3019612.3019878.

[40] M. Miettinen, S. Marchal, I. Hafeez, T. Frassetto, N. Asokan, A. R. Sadeghi, and S. Tarkoma. IoT Sentinel: Automated Device-Type Identification for Security Enforcement in IoT. In *Proceedings of the International Conference on Distributed Computing Systems (DCS)*, pages 2511–2514. IEEE, 2017. DOI 10.1109/ICDCS.2017.284.

[41] A. Mudgerikar, P. Sharma, and E. Bertino. E-Spion: A System-Level Intrusion Detection System for IoT Devices. In *Proceedings of the ACM Asia Conference on Computer and Communications Security (Asia CCS)*, pages 493–500. ACM Press, 2019. DOI 10.1145/3321705.3329857.

[42] M. Naveed, X. Zhou, S. Demetriou, X. Wang, and C. A. Gunter. Inside Job: Understanding and Mitigating the Threat of External Device Mis-Bonding on Android. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*. Internet Society, Feb. 2014. DOI 10.14722/ndss.2014.23097.

[43] NETSCOUT. DAWN OF THE TERRORBIT ERA. Technical report, Threat Intelligence Report, 2019.

[44] T. Peters. CRAWDAD dataset dartmouth/bluetooth-hci (v. 2021-03-29), Mar. 2021. Online at https://doi.org/10.15783/tjt0-b278.

[45] T. Peters, R. Lal, S. Varadarajan, P. Pappachan, and D. Kotz. BASTION-SGX: Bluetooth and Architectural Support for Trusted I/O on SGX. In *Proceedings of the Workshop on Hardware and Architectural Support for Security and Privacy (HASP)*, pages 1–9, Jun. 2018. DOI 10.1145/3214292.3214295.

[46] bluepy, 2021. Online at https://github.com/traviswpeters/bluepy.

[47] btsnoop, 2021. Online at https://github.com/traviswpeters/btsnoop.

[48] J. Ren, D. J. Dubois, D. Choffnes, A. M. Mandalari, R. Kolcun, and H. Haddadi. Information Exposure From Consumer IoT Devices: A Multidimensional, Network-Informed Measurement Approach. In *Proceedings of the Internet Measurement Conference (IMC)*, pages 267–279. ACM Press, Oct. 2019. DOI 10.1145/3355369.3355577.

[49] S. Satam, P. Satam, and S. Hariri. Multi-level Bluetooth Intrusion Detection System. In *Proceedings of IEEE/ACS International Conference on Computer Systems and Applications, AICCSA*, volume 2020-November. IEEE Computer Society, Nov.

2020. DOI 10.1109/AICCSA50499.2020.9316514.
[50] R. W. Shirey. Internet Security Glossary, Version 2. RFC 4949, Aug. 2013. DOI 10.17487/RFC4949.
[51] P. Shrestha and N. Saxena. Hacksaw: Biometric-Free Non-Stop Web Authentication in an Emerging World of Wearables. In *Proceedings of the ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*, pages 13–24, 2020. DOI 10.1145/3395351.3399366.
[52] D. Shukla, R. Kumar, A. Serwadda, and V. V. Phoha. Beware, Your Hands Reveal Your Secrets! In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, pages 904–917. ACM, 2014. DOI 10.1145/2660267.2660360.
[53] R. Sommer and V. Paxson. Outside the Closed World: On Using Machine Learning for Network Intrusion Detection. In *Proceedings of the IEEE Symposium on Security and Privacy (Oakland)*, pages 305–316, 2010. DOI 10.1109/SP.2010.25.
[54] J. Wampler, I. Martiny, and E. Wustrow. ExSpectre: Hiding Malware in Speculative Execution. In *Proceedings of the Network and Distributed Systems Security Symposium (NDSS)*, 2019. DOI 10.14722/ndss.2019.23409.
[55] K. Wang and S. J. Stolfo. *Anomalous Payload-Based Network Intrusion Detection.* Springer Berlin Heidelberg, 2004. DOI 10.1007/978-3-540-30143-1_11.
[56] C. Wressnegger, G. Schwenk, D. Arp, and K. Rieck. A Close Look on N-grams in Intrusion Detection: Anomaly Detection vs. Classification. In *Proceedings of the ACM Workshop on Artificial Intelligence and Security*, pages 67–76, 2013. DOI 10.1145/2517312.2517316.
[57] Y. Xu, W. Cui, and M. Peinado. Controlled-Channel Attacks: Deterministic Side Channels for Untrusted Operating Systems. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, pages 640–656, May 2015. DOI 10.1109/SP.2015.45.
[58] W. Zhang, Y. Meng, Y. Liu, X. Zhang, Y. Zhang, H. Zhu, and H.-J. Zhu. HoMonit: Monitoring Smart Home Apps from Encrypted Traffic. In *Proceedings of the Conference on Computer and Communications Security (CCS)*, 2018. DOI 10.1145/3243734.3243820.

A TESTBED & DATASET INSIGHTS

Here we present a few detailed insights into our smart device testbed and the new dataset presented in this paper.

A.1 Testbed Overview

In Figure 5 we include a photo of many of the devices from our testbed. Our testbed currently consists of 9 different device types, and 20 devices in total. Recall Table 1 and Table 2, which provide more details about the specific devices and apps used to collect data. Using our tedbed we produced a new dataset of more than 300 Bluetooth network traces [44].

Figure 5: Our collection of smart Bluetooth devices.

A.2 HCI-Related Features

In Figure 6 we provide an example of the distribution of HCI packet lengths based on attributes such as the direction of the packet (ingress/egress) and the type of packet. As discussed in Section 4.3, along with the contents of packets themselves, the differences observed between HCI packets (i.e., type, direction, length) are meaningful features for separation among VIA models.

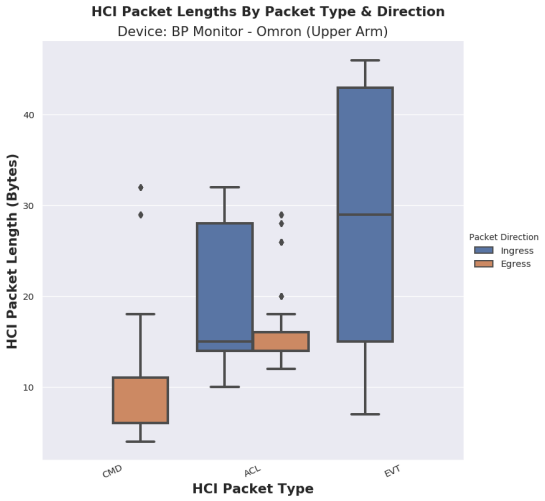

Figure 6: HCI packet lengths by type/direction.

A.3 Case Study

In Figure 7 we include a visualization of relatives frequencies of the *n*-grams computed from approximately 75 traces belonging to blood-pressure monitors. (Recall Section 6.3.1 where we discussed comparisons of these devices.)

The blood-pressure monitors present any interesting case study. In total there are 5 blood-pressure monitors (same type) from three different manufacturers, and some of these monitors take measurements at different locations on the body (upper arm and wrist). As a result, we can see roughly 5 distinct bands (1: 0-17, 2: 18-29, 3: 30-46, 4: 47-59, 5: 60-74). Each of these bands represent the *n*-grams computed from a distinct device. Notice that the intraband traces yield similar representations, which are distinct from traces belonging to other bands (devices); one exception seen here is the similarity within bands 2-3 and 4-5, which we posit is due to the fact that they are different models of devices (wrist vs. upper arm) made by the same manufacturer.

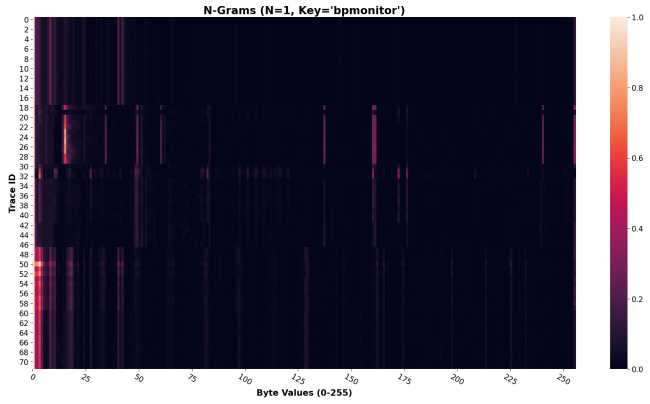

Figure 7: An example of *n*-grams computed from the traces in our dataset. These traces are selected from 5 different blood-pressure monitors made by 3 distinct manufacturers.