SniffMislead: Non-Intrusive Privacy Protection against Wireless Packet Sniffers in Smart Homes

Xuanyu Liu
Nanjing University
Nanjing, China
xuanyuliu@smail.nju.edu.cn

Qiang Zeng
University of South Carolina
Columbia, SC, USA
zeng1@cse.sc.edu

Xiaojiang Du
Temple University
Philadelphia, PA, USA
xjdu@temple.edu

Siva Likitha Valluru
University of South Carolina
Columbia, SC, USA
svalluru@email.sc.edu

Chenglong Fu
Temple University
Philadelphia, PA, USA
chenglong.fu@temple.edu

Xiao Fu
Nanjing University
Nanjing, China
fuxiao@nju.edu.cn

Bin Luo
Nanjing University
Nanjing, China
luobin@nju.edu.cn

ABSTRACT

With the booming deployment of smart homes, concerns about user privacy keep growing. Recent research has shown that encrypted wireless traffic of IoT devices can be exploited by packet-sniffing attacks to reveal users' privacy-sensitive information (e.g., the time when residents leave their home and go to work), which may be used to launch further attacks (e.g., a break-in). To address the growing concerns, we propose SniffMislead, a non-intrusive (i.e., without modifying IoT devices, hubs, or platforms) privacy-protecting approach, based on packet injection, against wireless packet sniffers. Instead of randomly injecting packets, which is ineffective against a smarter attacker, SniffMislead proposes the notion of *phantom users*, "people" who do not exist in the physical world. From an attacker's perspective, however, they are perceived as real users. SniffMislead places multiple phantom users in a smart home, which can effectively prevent an attacker from inferring useful information. We design a top-down approach to synthesize phantom users' behaviors, construct the sequence of decoy device events and commands, and then inject corresponding packets into the home. We show how SniffMislead ensures logical integrity and contextual consistency of injected packets, as well as how it makes a phantom user indistinguishable from a real user. Our evaluation results from a smart home testbed demonstrate that SniffMislead significantly reduces an attacker's privacy-inferring capabilities, bringing the accuracy from 94.8% down to 3.5%.

CCS CONCEPTS

• **Security and privacy → Privacy protections**; *Mobile and wireless security*.

KEYWORDS

Smart home, IoT device, wireless network, privacy, packet-sniffing attack

ACM Reference Format:
Xuanyu Liu, Qiang Zeng, Xiaojiang Du, Siva Likitha Valluru, Chenglong Fu, Xiao Fu, and Bin Luo. 2021. SniffMislead: Non-Intrusive Privacy Protection against Wireless Packet Sniffers in Smart Homes. In *24th International Symposium on Research in Attacks, Intrusions and Defenses (RAID '21), October 6–8, 2021, San Sebastian, Spain.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3471621.3471856

1 INTRODUCTION

The development of smart homes is flourishing, as inexpensive IoT devices become prevalent and IoT integration platforms, such as SmartThings, Google, and Amazon, emerge to enable interoperability between devices. As IoT devices usually communicate with hubs or clouds via wireless protocols such as ZigBee, BLE, Z-Wave, and WiFi, they are prone to wireless packet sniffing attacks.

Although encryption is used to protect wireless communication, recent works [1, 8, 9, 19, 57, 64, 79] have shown that privacy-sensitive information, such as device states and user behaviors, can be inferred from wireless packets. Packets of different IoT devices and their events and commands tend to show different patterns, called *packet-level signatures* [64, 79], from which a wireless packet sniffer can infer device types, actions, and states (e.g., `OPEN`/`CLOSED` of a contact sensor on a door), without knowing the encryption key. As IoT devices and users become increasingly coupled, it is feasible to infer user behaviors (e.g., the time she leaves home) [1, 18, 27, 41, 72, 73]. The privacy-sensitive information can facilitate further attacks, such as breaks-in when no one is home, crimes targeting users who live alone, selling personal information to criminals for profits, and blackmail [21, 25]. Such side-channel attacks are easy to launch, though difficult to detect.

Many countermeasures against these side-channel attacks have been proposed, which we group into two main categories. The first includes traffic shaping and packet padding to reduce the traffic variability among different devices [6, 9, 44, 64, 65, 71]. Countermeasures in this category suffer from the following limitations: 1) They require modifications of IoT firmware; 2) Changes in protocols (e.g., number of padding packets) may cause compatibility issues among IoT devices, hubs, and backend servers; 3) The redundant packets and padding bytes impose significant communication and energy overheads on resource-constrained IoT devices [9]. The second category of countermeasures is event spoofing via packet injection [1, 7, 64], which simulates *decoy* events of IoT devices to confuse an attacker from being able to distinguish between real and decoy events. However, a robust, efficient, and automated method for event spoofing is still an open research problem [7], especially since packet injection from IoT devices also requires modifications to IoT firmware and/or an extension to the protocols by adding a decoy-event flag [1].

Therefore, one may propose a *non-intrusive* solution, i.e., without modifying IoT firmware or underlying communication protocols, by implementing one of the aforementioned countermeasure approaches using a *stand-alone* device. However, this intuitive method will not work for the following reasons. First, it is unknown how a stand-alone device can predict upcoming traffic and then proactively perform shaping, and it is unlikely to pad encrypted packets, using a stand-alone device, without knowing the session keys. Second, the defense of injecting random packets from a stand-alone device is weak, as existing traffic analysis is resilient to noises [1, 39, 40]. Finally, the stand-alone device may purposely inject traffic to simulate some IoT events in order to fool attackers. However, for example, if a security defense injects traffic simulating motion-active events even after users have left their homes, then based on logical conflicts [7], an attacker can detect they are fake events, by using either causal relationship analysis [2, 10, 69] or context and integrity detection [26, 32, 37, 52]. As another example, assume the motion sensor regularly triggers light-on events in the presence of real users. Then based on missing light-on events, an attacker can infer the motion-active events are fake. Similarly, assuming a door is unlocked only when a presence sensor has been reported *on*, a fake door-unlock event can be easily recognized by attackers, presuming the presence-on state is false.

We propose a novel and effective defense, called SniffMislead, against wireless packet sniffers: a non-intrusive and resilient privacy-protecting solution via packet injection using a stand-alone device, without modifying IoT devices, hubs, platforms, or communication protocols, based on the notion of *phantom users*. Phantom users do not exist in the physical world but, from the perspective of an attacker, "live" in a smart home and act as real users would. The primary challenge is that, when analyzing real-time traffic, phantom users should remain indistinguishable from real users. To that end, we propose a top-down approach to simulate logically-sound phantom users by first designing their behaviors. Next, for each behavior, we construct a sequence of device events and commands consistent with the home context (e.g., a phantom user should not lock a door if it is already locked; the phantom user should not turn off the light if someone is still in the room). Finally, corresponding to each event and command of an IoT device, we inject decoy packets into the target smart home network.[1]

Behaviors of phantom users are designed to be as different from each other (e.g., one phantom user is cooking, and another watching TV) and involve as many diverse IoT devices as possible at any given moment, such that attackers are not confident whether an inferred device state is real or not. Behaviors of phantom users are also designed to be different from those of real users, so attackers are not sure whether an inferred user behavior or home state is real or due to a phantom user. By simulating phantom users, SniffMislead ensures logical integrity (i.e., decoy device events can correctly compose behaviors of a phantom user) and contextual consistency (i.e., a phantom user inside the "multi-user" smart home do not introduce conflicts) of injected packets and simulated decoy device events, preventing attackers from making reliable inferences about *device states or behaviors of real users* (e.g., the attacker observes that a user is watching TV, when, in fact, no one is home).

Through this work, we make the following contributions:

- Unlike existing works, SniffMislead is a stand-alone and non-intrusive countermeasure against wireless packet sniffers. It works independently and causes no changes to IoT devices or the underlying communication protocols. It is a plug-and-play solution, i.e., requiring no additional configuration efforts from users.
- We propose the notion of *phantom users* and use a top-down approach to synthesize their behaviors and inject corresponding packets into the target smart home, ensuring that phantom users remain indistinguishable from real users.
- We build a prototype of SniffMislead and evaluate it in a real-world smart home. Our results show that SniffMislead significantly undermines a wireless sniffing attacker's capability (from 94.8% to 3.5%) of inferring user behaviors, as he is unsure whether his inference is coming from a phantom or real user.

The remainder of this paper is structured as follows. Section 2 outlines related work. Section 3 discusses the threat model we adopted, our goals, and anticipated design challenges. Section 4 presents the design and workflow of SniffMislead. Section 5 discusses how to learn the needed information from a smart home. Section 6 discusses how to inject decoy packets that simulate phantom users. Section 7 presents our experimental setup. Section 8 summarizes our evaluation results. Section 9 addresses limitations of SniffMislead and discusses potential follow-up studies. The paper is concluded in Section 10.

2 RELATED WORK

Side-Channel Analysis Based on Sniffed Packets. A growing body of work uses network traffic side-channel analysis to infer states of IoT devices. It relies on the limited-purpose nature of IoT devices, and there is a one-to-one mapping between traffic patterns and device events. A series of papers by Apthorpe et al. [6–9] use traffic volume-based signatures to infer device events. HomeSnitch [42] identifies IoT events based on an observation that the

[1]For the sake of brevity, both events and commands of IoT devices will be collectively referred to as *device events* for the rest of this paper.

client (i.e., the IoT device) and server take turns in a request-reply communication style. HoMonit [79] uses side-channel information on encrypted wireless packets to monitor the misbehavior of smart apps. It was found that device events can be inferred using packet size and direction, which agrees with the observation from Ping-Pong [64]. To infer sensitive information based on side-channel information, some methods rely on machine learning [1, 36, 54, 55], while others use statistical analysis [1, 19, 42, 46].

Existing Defenses. Existing IoT privacy-protection work can be divided into three main categories: privacy protection against 1) IoT apps [11, 12, 25, 35, 61, 77], 2) IoT platforms [14, 15, 72], and 3) traffic sniffers [1, 7, 9, 20]. For example, FlowFence [25] protects sensitive data from being leaked via IoT apps by enforcing a data flow control mechanism. PFirewall [15] is a semantics-aware customizable data flow control system for smart homes to protect user privacy from IoT platforms, by filtering and obfuscating data generated by IoT devices. SniffMislead belongs to the third category. Several approaches have been proposed to obfuscate network traffic to defend against network traffic side-channel analysis. Packet padding [23, 44, 71] adds dummy bytes to each packet. Packets can be padded to a fixed length or with a random number of bytes to confuse inference methods that rely on individual packet lengths. Traffic shaping [6, 20, 65] purposely delays packets to a fixed rate. It can confuse inference methods that rely on packet inter-arrival times and volume over time. Packet padding and traffic shaping require changes of IoT firmware/protocols, cause high overhead to devices; e.g., traffic shaping may delay packets too much, harming user experience. Traffic injection and event spoofing [1, 7, 64] add dummy packets or decoy events to make it difficult for an attacker to distinguish which ones are real. These countermeasures, without a global view of events at the protected home, are ineffective against advanced methods, such as traffic analysis [1, 39, 40], causal relationship analysis [2, 10, 69], and context and integrity detection [26, 32, 37, 52]. Plus, they also require modifications to IoT firmware and/or an extension to the protocols by adding a decoy-event flag [1], because they use IoT devices to add dummy packets or decoy events.

3 THREAT MODEL, GOALS, AND CHALLENGES

In this section, we first present the threat model, which includes the smart home environment and attackers that we consider. Next, we describe the goals and design challenges of SniffMislead.

3.1 Threat Model

As shown in Figure 1, an IoT device uses a wireless connection to communicate with either a hub or router, both of which are prone to packet sniffing. We consider a passive attacker similar to recent works [1, 8, 9, 64], who can remotely control a wireless packet sniffer (e.g., based on a compromised IoT device), located within the wireless range of a target smart home, to eavesdrop on wireless traffic inside the house. The attacker is *not physically close* to the home and thus, for instance, cannot see who is entering or leaving the home; none of existing countermeasures, such as traffic shaping and packet padding, are able to fool such attackers. SniffMislead is deployed inside a smart home and injects decoy packets into the wireless network of the home, to confuse and mislead the attacker.

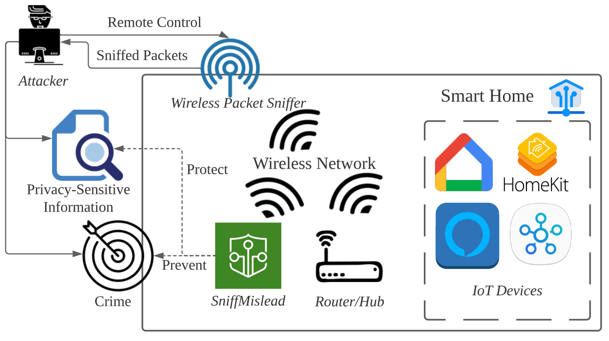

Figure 1: Attackers versus SniffMislead in a smart home environment.

Attackers' Capabilities. Major IoT communication protocols, such as ZigBee [24], Z-Wave [51], and BLE [49], require mandatory encryption. Besides, most WiFi-based IoT devices use encryption. For example, the survey [5] lists the devices that use encryption in Table II; among 45 devices evaluated in the paper, only three use unencrypted wireless communications. Although most wireless packets of smart home devices are encrypted, traffic metadata (e.g., timestamps, lengths, and directions) is still available to attackers. Attackers also have access to unencrypted packet headers, which are used to extract valuable information such as Network and MAC addresses. Via side-channel analysis, attackers can use these data to infer privacy-sensitive information about the target home, such as IoT device types, device states, and user behaviors. Attackers can leverage various analysis techniques to identify packets injected randomly [1, 39, 40] and spoofed events that violate causal relationship [2, 7, 10, 69], context and integrity [26, 32, 37, 52].

For the small number of IoT devices that send unencrypted packets, attackers can see the meaningful information about an IoT device from the payload directly and thus need not use any side-channel analysis method. SniffMislead is designed to defend against side-channel analysis on encrypted packets. Countermeasures against information leakage from unencrypted packets are out of the scope of this work.

Attackers' Limitations. 1) Attackers are passive; i.e., they sniff quietly and do not proactively inject packets in the target smart home. It is straightforward to extend SniffMislead to fight active attacks that inject IoT packets, e.g., by comparing the received packets against the ground truth of IoT device states based on log from IoT platforms [69]. Such an extension falls beyond the contributions of this work. 2) We focus on side-channel attacks, so attacks on encryption and communication protocols are out of scope of this paper. Therefore, attackers cannot break the encryption and claim access to the clear-text communication.

Attackers' Goals. Attackers have three main goals: 1) They want to infer states of IoT devices. For example, is the window open during night? 2) They want to infer user behaviors and habits inside the target home. For instance, when does the user go to bed? 3) They want to infer privacy- and security-sensitive states of the home overall. For example, has everyone left the house?

A cyber-attacker can steal the information from a large number of homes at scale and sell it to criminals, who may leverage it for, e.g., breaks-in, attacking users who live alone, or blackmailing.

3.2 Goals and Challenges

Our Goals. We propose SniffMislead, an effective solution for privacy protection, against wireless packet sniffers in smart homes with the following **goals** in mind: 1) Being non-intrusive and easy to deploy; 2) Bringing no side effects to the target home (e.g., does not disrupt the normal operations of IoT devices); 3) Dramatically impairing the capability of attackers in attaining their goals.

We clarify that SniffMislead is not to prevent attackers from inferring information, since real wireless traffic can still be sniffed by attackers. Instead, due to the added phantom users, many resulting fake events are injected. As a result, given an inferred event, the attacker has a very *low* confidence whether it is a real or injected one. Similarly, given an inferred behavior of a user, the attacker does not know whether the user is a real or phantom user.

Challenges. As discussed previously, it is challenging to perform traffic shaping and packet padding using a stand-alone device. Packet injection and event spoofing, injected randomly or intuitively, are also inefficient. Real device events, triggered by user behaviors, are self-consistent with rich semantics and full context. Decoy events, on the other hand, with a lack of logical integrity, may suffer from being distinguished from logically-complete ones. In addition, some decoy events are unable to "change" the semantics of the user behaviors or context of the target smart home. Therefore, they fail to prevent attackers from inferring correct user behaviors (e.g., a decoy light-off event in the living room may not change the inference result that a real user is watching TV). Thus, a comprehensive approach is essential to make injected packets and simulated events indistinguishable from real ones and retain enough logicality and capacity to change the target smart home's semantics and context. Placing logically-sound phantom users, therefore, is a feasible and much-needed choice, though it is not trivial to do so. We need to understand what behaviors compose a phantom user in the target home, what device events compose a behavior, and how to inject packets to simulate device events.

In order for SniffMislead to work independently without configuration effort from the user, it has to generate a packet-injection policy on its own. The device list, automation rules of device events, and device-event-level features of user behaviors[2] are unknown to SniffMislead since they dynamically vary from home to home. A fixed policy is therefore insufficient. The policy should be adaptive to ensure that phantom users can pass off as humans. Therefore, in order to learn from the target smart home and use its unique features to generate resilient policies for placing phantom users, SniffMislead would need to be implemented from scratch. SniffMislead also has been programmed to avoid logical conflicts between phantom and real users (e.g., a phantom user wants to turn off the light while a real user wishes to keep it on).

[2]We define the *device-event-level feature* of a user behavior as the pattern indicating how device events can construct an ongoing activity.

Causing no change to the target smart home means that real wireless traffic from IoT devices cannot be blocked or modified. Plus, IoT devices at a home are subject to changes, e.g., adding or removing devices. SniffMislead needs to respond to those changes to preserve contextual consistency.

4 SYSTEM OVERVIEW

In this section, we discuss how we achieve the targeted goals and overcome the proposed design challenges. We first present the features of SniffMislead and then describe its workflow in detail.

4.1 Features of SniffMislead

SniffMislead has the following features:

- It is a non-intrusive solution that does not require any modifications to IoT devices, hubs, platforms, or communication protocols. It is a plug-and-play solution that uses a stand-alone device. It works independently without needing any configuration effort from users or prior knowledge of the target home (e.g., attributes of smart devices).
- It is an automatic wireless packet injection and event spoofing tool. It uses a top-down approach (i.e., behaviors $\rightarrow$ device events $\rightarrow$ wireless packets) to place phantom users in a home, which, from an attacker's perspective, would seem as if there are "real" users in the home. By ensuring logical integrity and contextual consistency of injected packets and simulated decoy device events, it is difficult for attackers to distinguish phantom users from real ones.
- To generate comprehensive, adaptive policies for placing phantom users, SniffMislead learns required information (i.e., traffic patterns of smart devices, device-event-level features of user behaviors, associations among behaviors, and real users' daily routines) on its own by analyzing encrypted wireless traffic of the target smart home. The policies, along with any changes in the target smart home, are continuously and dynamically updated.
- SniffMislead injects packets only based on the target home's traffic pattern, independent of underlying physical layers, device types, layout, or house structures. It can simultaneously handle multiple network protocols by including different packet sniffers and transmitters.

4.2 Workflow of SniffMislead

SniffMislead has two modules: 1) *Smart Home Learning Module* (Section 5), and 2) *User Privacy Protection Module* (Section 6).

Smart Home Learning Module. This module learns required information from the target smart home to prepare for policy generation. It consists of the following four steps, shown in Figure 2: 1) Collecting a training set that contains wireless network traffic generated by IoT devices (Section 5.1); 2) Extracting packet-level signatures of device events from the training set (Section 5.2); 3) Extracting device-event-level features of user behaviors (Section 5.3), using the output from Step 2; and 4) Obtaining logical associations among behaviors and real users' daily routines (Section 5.4). Learned information will be stored in a database. SniffMislead continuously inspects changed traffic patterns caused by changes in user behaviors and IoT devices, and updates its database.

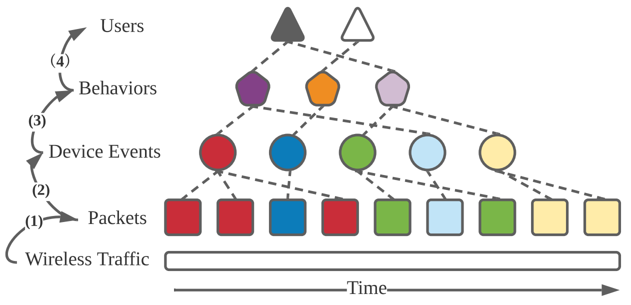

Figure 2: Four steps of Smart Home Learning: (1) collecting data from wireless traffic, (2) extracting packet-level signatures of device events, (3) extracting device-event-level features of behaviors, and (4) finding associations among behaviors and daily routines of real users.

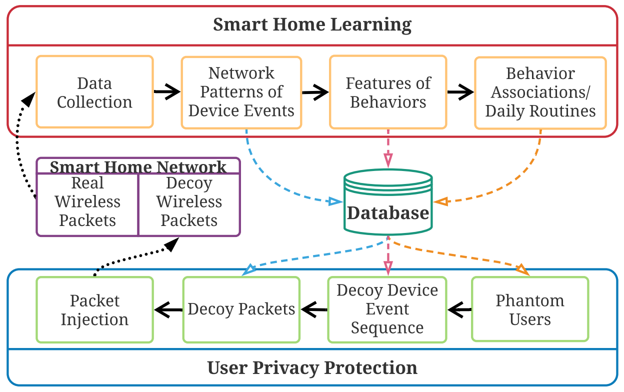

Figure 3: Workflow of SniffMislead.

User Privacy Protection Module. This module generates policies based on information learned from the *Smart Home Learning* Module above and decides on how to use a top-down approach to simulate phantom users. Behaviors for phantom users are generated every day, based on logical associations among behaviors and real users' daily routines (Section 6.1). Then, for each behavior of a phantom user, the corresponding device event sequence is generated, based on the device-event-level features of this behavior (Section 6.2). Finally, based on packet-level signatures of device events, decoy packets (corresponding to the device event sequence) are created (Section 6.3).

Figure 3 shows the building blocks of the two modules and the workflow of SniffMislead. We use an example to briefly discuss how SniffMislead protects user privacy. In a typical smart home with common IoT devices (e.g., motion sensors, smart outlets, lights), SniffMislead would create phantom users and set their behaviors (e.g., waking up at 6:50 a.m., having breakfast at 7 a.m., watching TV at 8 a.m., and so on). When it is time for a phantom user's behavior to happen (i.e., it is 7 a.m. to have breakfast), this behavior is converted to a sequence of device events (e.g., OPEN of a contact sensor; ACTIVE of a motion sensor; ON of the kitchen lights; ON of an outlet; ON of the oven). Finally, based on each device event's unique packet-level signature, wireless packets are generated for each device using a packet transmitter. When injected decoy packets are captured by a wireless packet sniffer and mapped to device events and user behaviors by the attacker, it will be difficult for the attacker to reliably infer device states or behaviors of real users (e.g., in fact, no one is having breakfast at 7 a.m.).

5 SMART HOME LEARNING

In this section, we discuss how SniffMislead learns required information from the target smart home by sniffing its encrypted wireless traffic, with no configuration effort from users or prior knowledge of the smart home.

There may exist correlations [16, 17, 28, 52] between IoT devices in the smart home, in the form of co-present or temporally-related device events. These device events cause network communications among IoT devices, hubs, and routers. These correlations can be attributed to three correlation channels: 1) Automation rules (i.e., smart apps) [13, 15], which directly determine trigger-condition-action relationships among IoT devices (e.g., a button is configured to flip a light); 2) Physical interactions [22], i.e., changes in physical properties (e.g., humidity, motion, and smoke) may cause actions of nearby devices to respond to these changes (e.g., turning on a smart light can affect an illuminance sensor nearby; the rise of temperature could be captured by both an air conditioner and a temperature sensor, resulting in two consecutive events); 3) User behaviors [29], i.e., while user behaviors impose changes on devices, device states also reflect user behaviors (e.g., when a user returns home, there should be consecutive events, such as OPEN of a contact sensor for door opening and ACTIVE of a motion sensor for the user's moving).

SniffMislead learns these correlations from network messages (i.e., wireless traffic) of the target smart home, and exact network-level and device-level features, which are used for injecting packets of decoy device events.

5.1 Step 1: Data Collection

The first step is to form a training set by collecting related wireless packets from smart home network traffic. Multiple packet sniffers, for different protocols, automate this procedure using shell scripts. Since the network identifier is usually unique and unencrypted and can be seen from the packet header, it can be used to identify devices [38, 50, 64, 79]. For example, the Network Address in the ZigBee packet header can be used to distinguish ZigBee devices, as it is created and assigned when a device joins the smart home network and remains unchanged until it leaves the current network and joins another network.

Each found network identifier is treated as a separate device, with a device identifier assigned to it, denoted as <$DeviceIdentifier$, $NetworkIdentifier$>. Collected packets are grouped by devices, and SniffMislead uses $DeviceIdentifier$ to distinguish IoT devices in the target smart home. Whenever a smart home user adds or deletes a device, the corresponding device list and existing network identifier are updated. SniffMislead continuously responds to these changes and update its database.

Since packets triggered by device events always have encrypted payloads, those without payloads and/or unrelated to users' operations and behaviors (e.g., beacon packets that are mainly used to acknowledge data transmission and maintain established connections) are discarded. For re-transmitted packets, SniffMislead only uses the first one and discards the others.

5.2 Step 2: Extracting Network Patterns of Device Events

IoT device events usually have unique network patterns, summarized as packet-level signatures [64, 79]. Side-channel data, such as lengths, rates, and directions of packets, can identify device events. A *packet* is defined as a tuple of (*timestamp*, *length*, *direction*). Each device event triggers a time-ordered sequence of wireless packets. SniffMislead will then establish a one-to-one mapping between a device event e_i and its network pattern $Pattern_{e_i}$.

A *burst*, presented as $b = \{packet_1, packet_2, \cdots, packet_n\}$, is a sequence of network packets from one device, where the time interval between any two consecutive packets is less than a predetermined threshold β [59, 79]. Grouped packets from Step 1 are partitioned into a set of bursts. If the packet directions match, *Levenshtein Ratio* [43, 74] between these bursts is calculated to indicate the level of similarity. A high Levenshtein Ratio means a high similarity. It is validated that the average Levenshtein Ratio is stable for device events of the same type (i.e., 0.98 for ZigBee and 0.98 for Z-Wave) and distinguishable for events of different types (i.e., 0.17 for ZigBee and 0.25 for Z-Wave) [79]. Based on our empirical observations from our testbed, we select 0.8 as the Levenshtein Ratio threshold: γ. Bursts, whose Levenshtein Ratio is greater than γ, are grouped together: $B = \{b_1, b_2, \cdots, b_n\}$. Then the network pattern of a device event e is calculated as:

$$Pattern_e = \arg\max_{b_i \in B} \sum_{\forall b_j \in B} \mathcal{R}(b_i, b_j) \tag{1}$$

where $\mathcal{R}$ is the Levenshtein Ratio of b_i and b_j.

A device usually has multiple states. Transitions among states, caused by device events, usually correspond to some rules (e.g., `ON` $\longrightarrow$ `OFF` $\longrightarrow$ `ON` for a light). After extracting network patterns of device events, the traffic stream in the training set is converted into a device event stream. Based on that, SniffMislead can then build finite state machines for each device to indicate transition rules, which are used later for simulating decoy device events. Each smart device may have one or more finite state machines.

5.3 Step 3: Extracting Features of Behaviors

A behavior usually triggers specific device events following a feature (e.g., involved device events and their occurrence time), denoted as a device-event-level feature, which, in turn, can also be used to predict an ongoing behavior [1, 9, 19]. To generate appropriate behaviors of phantom users, SniffMislead needs to find device-event-level features of behaviors in the target smart home. However, it is challenging to extract the feature of each behavior. Based on our observations, due to variability in the trigger of device events between different instances of the same behavior, one or a few devices cannot determine a specific behavior. Device events also often get multiplexed from two or more behaviors, due to one or multiple users concurrently undertaking many behaviors.

To tackle the challenges mentioned above, SniffMislead first segments the device event stream from the training set, with the notion that each segment represents a behavior accurately (Section 5.3.1). Next, based on these segments, device-event-level features of behaviors are extracted (Section 5.3.2). SniffMislead does not need to know the specific behavior. It simply assigns each behavior a unique identifier and maps it to its feature, denoted as <*BehaviorIdentifier*, *BehaviorFeature*>. SniffMislead utilizes the features of behaviors to generate decoy device events and simulate a decoy behavior.

5.3.1 *Segmentation of Device Event Streams.*

A device event stream is partitioned into a set of segments $S = \{s_1, s_2, \cdots, s_i\}$, where s_i is a segment, and denoted as $s_i = \{e_1, e_2, \cdots, e_j\}$, and e_j is a device event. The partition is performed based on three factors: 1) the time interval among device events, 2) the proximity among device events, and 3) the frequency of device events' occurrence, all three of which have proved to be effective for de-multiplexing device events [33].

Time-Interval-Based Segmentation. The goal of this step is to separate temporally-distinct behaviors. We introduce two types of time intervals: inter-event time interval t_e and inter-segment time interval t_s. We observe that for most behaviors, inter-segment time intervals are significantly larger than inter-event time intervals. When applying the segmentation criterion, a non-negative temporal threshold τ is required. If device events belong to a segment, any inter-event time interval between them is less than τ, and any inter-segment time interval is longer than τ:

$$\max(t_e^i) \le \tau \le \min(t_s^i)$$

where t_e^i is the inter-event time interval between event e_{i-1} and e_i; t_s^i is the inter-segment time interval between segment s_{i-1} and s_i.

Proximity-Based Segmentation. The probability of device events from a neighboring behavior being wrongly assigned to a segment is exponentially higher than the probability of events that come from behaviors further apart in time [33]. If two device event sets E_i and E_j in neighboring segments belong to the same behavior and should be re-partitioned into one segment, the following conditions must be met:

$$\forall E_k \in \mathcal{L}, \max(P_s^{jk}) < P_n^{ij},$$

where P_n^{ij} is the number of occurrences of E_i and E_j in neighboring segments; P_s^{jk} is the number of occurrences of two device event set E_j and E_k in the same segment; $\mathcal{L}$ is the set of all the device event sets that are partitioned into the same segment with E_j.

Frequency-Based Segmentation. We observe that when behaviors get multiplexed together within the same segment, the frequency of device triggering is different for different behaviors. Considering there are two device event sets E_i and E_j in one segment, if E_j has a stronger association with E_i and they are considered to belong to the same behavior, the following conditions must be met:

$$\forall E_k \in \mathcal{M}, \max(F_{jk}) < F_{ij},$$

where F_{ij} is the the number of occurrences of E_i and E_j in the same segment; F_{jk} is the number of occurrences of E_j and E_k in the same segment; $\mathcal{M}$ is the set of all the device event sets that found in the same segment with E_j.

5.3.2 Extracting Deterministic and Non-Deterministic Attributes of Behaviors. As mentioned previously, there are variations for different instances of the same behavior. Similar segments should be separated from dissimilar ones. The behavior feature is denoted as a tuple, $Feature = (Event, Time)$, where $Event$ indicates the involved device events and $Time$ is the events' temporal information. Segments are converted into these tuples first. Since we do not know the number of behaviors, we adopt the K-means algorithm with a gap statistic [62]. Similar segments are clustered into a group. Then, features of behaviors are extracted from each group.

Different instances of a particular behavior may have an inherent pattern. The feature of a behavior consists of two parts: deterministic attributes and non-deterministic attributes. For a behavior, some attributes may remain unchanged. For example, for a cooking-related behavior, the kitchen light is turned on after a motion sensor detects the presence of a user nearby. This attribute is deterministic, i.e., it can be found every time a cooking behavior takes place. Besides deterministic attributes, there are also varying attributes. For example, a microwave, refrigerator, or tea kettle is not always used during cooking. These attributes are non-deterministic. They represent changes and randomness while deterministic attributes are a behavior's inherent features. SniffMislead extracts both attributes from the group of segments. The feature of a behavior b can be represented as:

$$Feature_b = \mathcal{F}(b) = f_d \cup f_n, \tag{2}$$

where f_d and f_n are deterministic and non-deterministic attributes respectively.

5.4 Step 4: Extracting Behavior Associations

After extracting features of behaviors, the device event stream in the training set is converted into a behavior stream. The behaviors of real smart home users typically follow some patterns, denoted as daily routines. To create credible phantom users and make them different from real users, we need to extract those patterns based on the behavior stream. We consider three major associations among behaviors in a target home:

Behavior-to-Time Associations. Behavior-to-time associations represent relationships between behaviors and their occurrence time, concerning the hour of the day, day, week, and month. According to [73], we select a one-hour time slice, which sufficiently captures associations, minimizes the number of segments created, and prevents over-fitting by ensuring a maximum of twenty-four clusters for a day. The probability of a behavior b_i happening at time $t_i = \{h_i, d_i, w_i, m_i\}$:

$$\mathcal{P}_t(b_i, t_i) = p_h(bi, h_i) \cdot p_d(b_i, d_i) \cdot p_w(b_i, w_i) \cdot p_m(b_i, m_i) \tag{3}$$

where h_i is hour, d_i is day, w_i is week, m_i is month, whereas p_h, p_d, p_w, and p_m denote the probability of the behavior happens during the hour, day, week, and month, respectively.

Behavior-to-Behavior Association. When one behavior takes place, others may follow, e.g., eating breakfast after waking up. We use a directed acyclic graph (DAG) to represent this association, where nodes represent behaviors and edges indicate probabilistic dependencies. The graph provides a compact representation of a joint probability distribution, which describes that the probability of one behavior, b_i, is dependent on the probabilities of previous behaviors - $parents(b_i)$:

$$\mathcal{P}_b(b_i) = \prod_{j=1}^{i} p(b_j | parents(b_j)) \tag{4}$$

Temporal Relationships among Behaviors. There exist temporal relationships among behaviors. Behaviors may occur one after the other, concurrently or interleaved with each other, modeled as sequential, concurrent, and interleaved behaviors [41], respectively. Sequential behaviors are those that take place when one behavior is performed before or after another. Concurrent behaviors share the same time intervals, either fully or partially, and occur either when one user performs two different behaviors simultaneously or when multiple users perform behaviors simultaneously. Interleaved behaviors indicate that a behavior may occur within another behavior that is long and complicated. The temporal relationship between two behaviors may not be unique. For two behaviors, b_i and b_j, their temporal relationships are represented as:

$$\mathcal{T}(b_i, b_j) = \acute{\mathcal{T}}(t_s^i, t_e^i, t_s^j, t_e^j), \tag{5}$$

where t_s^i and t_e^i are the start and end time of behavior b_i, and t_s^j and t_e^j are the start and end time of behavior b_j. Temporal relationships are, in fact, the results of comparison of t_s^i, t_e^i, t_s^j, and t_e^j.

6 USER PRIVACY PROTECTION

After obtaining the required information from the target smart home, SniffMislead starts to generate policies that decide how to protect user privacy. In this section, we discuss how to use a top-down approach to simulate phantom users. SniffMislead first generates behaviors of phantom users. Then, for each behavior, it constructs the sequence of device events. Finally, corresponding to each device event, decoy packets are injected into the target home.

6.1 Generating Behaviors for Phantom Users

We define a behavior pattern, BP, as a set of behaviors that describe what a phantom user do in a day, denoted as: $BP = \{< b_0, t_0 >, < b_1, t_1 >, < b_2, t_2 >, \cdots, < b_n, b_n >\}$, where b_i is a behavior and $t_i =< ot_i, dt_i >$ is its temporal information including occurrence time ot_i and duration time dt_i. For example, behaviors such as "7 a.m., 5 minutes for getting up; 11 a.m., 1 hour for cooking; 2 p.m., 2 hours for working; 7 p.m., 3 hours for watching TV" are part of one behavior pattern.

A phantom user that has fixed behavior patterns would cause attackers suspicious. To overcome this, SniffMislead generates dynamic behaviors for phantom users every day. Based on the learned three behavior associations (Section 5.4), a behavior b_i of a phantom user, along with its time t_i, is determined using the function $\mathcal{A}$:

$$< b_i, t_i >= \mathcal{A}(\mathcal{P}_t, \mathcal{P}_b, \mathcal{T}, par_i, e_i), \tag{6}$$

where $\mathcal{P}_t$ and $\mathcal{P}_b$ are probability functions of behavior occurrence (Equations 3 and 4); $\mathcal{T}$ is temporal relationships among behaviors (Equation 5); par_i indicates behaviors of a phantom user that already happened before moment ot_i; e_i is a random noise, which brings some randomness.

A concern of dynamic behavior generation is the possibility of overfitting the model; if there is a high similarity between real and phantom users, attackers may still be able to infer a real user's behaviors. Hence, phantom users' behavior patterns should be different from those of real users. Since the behavior-to-time and behavior-to-behavior associations of real users (i.e., $\mathcal{P}_t$ and $\mathcal{P}_b$) can be used to predict real users' behaviors, SniffMislead determines the occurrence time of a phantom user's behavior, to ensure that real and phantom users are doing different things. Based on behavior-to-behavior associations and temporal relationships among behaviors (i.e., $\mathcal{P}_b$ and $\mathcal{T}$), the next behavior of a phantom user is determined according to previous behaviors and their duration. As a result, assigned behaviors to a phantom user are logically-sound, probabilistic, and unpredictable. Behavior patterns of phantom users are designed to be as different from each other as possible, to involve more smart devices at any given moment. The function of generating a behavior pattern BP for a phantom user in a day is summarized as:

$$BP = \mathcal{B}(\mathcal{A}, < b_0, t_0 >), \tag{7}$$

where $\mathcal{A}$ is the function of generating a behavior b_i along with its time t_i (Equation 6); b_0 is the first behavior in a day, i.e., waking up, with time t_0 being randomly determined from a time window.

The algorithm of $\mathcal{B}$ is shown in Algorithm 1.

Algorithm 1 Daily Behavior Generation for a Phantom User.

```
Input: $\mathcal{P}_t, \mathcal{P}_b, \mathcal{T}, < b_0, t_0 >$;
Output: Daily behavior pattern $BP = \{< b_0, t_0 >, < b_1, t_1 >, \cdots, < b_n, t_n >\}$;
  initial $< b_0, t_0 >$, add $< b_0, t_0 >$ to $BP$;
  repeat
      $par_i =< b_0, t_0 >, < b_1, t_1 >, \cdots, < b_{i-1}, t_{i-1} >$;
      $< b_i, t_i >= \mathcal{A}(\mathcal{P}_t, \mathcal{P}_b, \mathcal{T}, par_i, e_i)$;
      add $< b_i, t_i >$ to $BP$;
  until end of the day
```

SniffMislead does not have a limit of the number of phantom users to be simulated in a smart home. Intuitively, the accuracy of attackers inferring user behaviors decreases if the number of phantom users increases. Nevertheless, on the other hand, it becomes unreal to have too many phantom users because each home realistically has a limited number of residents from the perspective of a bystander. Therefore, it is better to strike a balance between the two aspects. After SniffMislead learns the attributes of the target smart home, it can train a behavior inference model. After deployment of SniffMislead, it attempts to increase the number of phantom users (i.e., adding one per every day, up to an upper limit) and form a dataset (i.e., combining real and decoy events). Then, SniffMislead uses its inference model on the dataset to measure the accuracy of inferring real behaviors. We define the accuracy here as the ratio of correctly-inferred behaviors to all behaviors of the same type. When the accuracy decreases to any extent less than the low rate, λ, the current number of phantom users is deemed appropriate. The value λ is configurable.

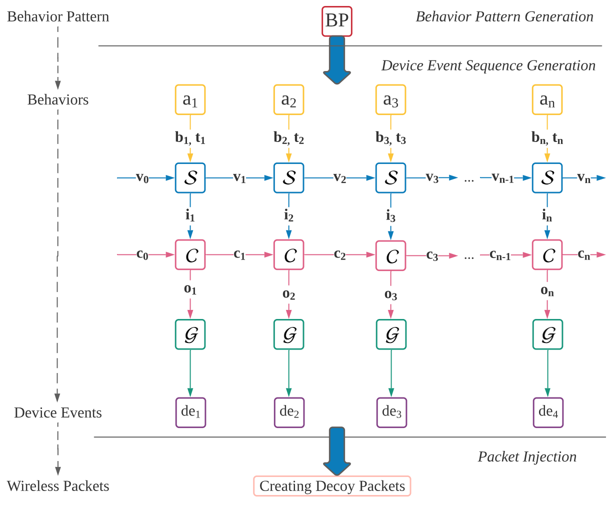

Figure 4: Workflow of generating device events based on behavior pattern BP and its individual behavior-time pairs, $a_1, \cdots, a_n$. The outputs are sequences of device events, $de_1, \cdots, de_n$, used for creating packets for injection. Functions $\mathcal{S}$, $\mathcal{C}$, and $\mathcal{G}$ are intermediate steps.

6.2 Generating Device Event Sequence

After behavior patterns for phantom users are generated, SniffMislead needs to set the sequence of each behavior's device events. As device-event-level features of behaviors have been learned (Section 5.3.2), SniffMislead generates device event sequences based on deterministic attributes first and then non-deterministic ones.

However, behaviors are usually not independent. Previous behaviors may influence the latter ones (e.g., a phantom may need to turn off the light when leaving home if the light has been turned on in a previous behavior). Therefore, when generating device events for current behavior, information about previous behaviors needs consideration as well. This helps create reasonable device events and enables a more natural transition between two neighboring behaviors. Finite state machines of smart devices (Section 5.2) are also used to ensure the correctness of state changes. The generated behavior pattern of a phantom user is parsed first; for each pair, $a_n =< b_n, t_n >$, the function of generating device event sequence for the behavior b_n and its time t_n is summarized below:

$$i_n, v_n = \mathcal{S}(\mathcal{F}(b_n), t_n, v_{n-1}, e_n), \tag{8}$$

where $\mathcal{F}(b_n)$ is the device-event-level feature of b_n (Equation 2); t_n is the time information about b_n; v_{n-1} includes information about the previous behavior b_{n-1}, and e_n is a random noise. The function $\mathcal{S}$ has two outputs: i_n, an initial version of a sequence of device events, and v_n, information about current behavior that may influence the next behavior.

The contextual information in a smart home should be consistent with each other. Hence, simulated decoy events of a device should not cause any conflict among the home users (both real and phantom), in the same smart home setting. E.g., a phantom user turns off the light while another phantom user is "in" the room.

To this end, SniffMislead needs to continuously maintain the *decoy* context of the target smart home, which, from an attacker's perspective, includes both real and phantom users. SniffMislead keeps monitoring real home context (i.e., real states of smart devices and real user behaviors) by monitoring real wireless traffic from the target smart home. The decoy home context is generated by combining the real home context with the influence of phantom user behaviors. The function to ensure context consistency and integrity of device event sequence is summarized below:

$$o_n, c_n = C(i_n, c_{n-1}), \tag{9}$$

where i_n is the output from function $\mathcal{S}$ (Equation 8); c_{n-1} is the context information; o_n is the output to help generate final version of the device event sequence; c_n is the updated context.

The function to generate the final device event sequence, de_n, for behavior-time pair $a_n =< b_n, t_n >$ is summarized below:

$$de_n = \mathcal{G}(o_n), \tag{10}$$

where o_n is the output from function C (Equation 9).

The overall workflow of generating device event sequence for a phantom user's behavior pattern is given in Figure 4, and the algorithm is presented in Algorithm 2.

Algorithm 2 Device Event Generation for a Phantom User

```
Input: Behavior pattern $BP$ for a phantom user;
Output: Device event sequence $DE = \{de_1, de_2, \cdots, de_n\}$;
  parse $BP$ to a list of behavior-time pairs: $< b_0, t_0 >, < b_1, t_1 >, \cdots, < b_m, t_m >$;
  initial $v_0$, $c_0$ based on $< b_0, t_0 >$; $n = 1$;
  repeat
      $i_n, v_n = \mathcal{S}(\mathcal{F}(b_n), t_n, v_{n-1}, e_n)$;
      $o_n, c_n = C(i_n, c_{n-1})$;
      $de_n = \mathcal{G}(o_n)$;
      add $de_n$ to $DE$;
      $n = n + 1$;
  until $n > m$
```

6.3 Packet Injection

After the device event sequence is generated, SniffMislead injects decoy packets, corresponding to each device event. In order to fool attackers, SniffMislead needs to make sure that there is no difference between injected packets and real ones. Given a device event, SniffMislead obtains the corresponding communication protocol, network identifier, and traffic pattern from the network pattern database (Sections 5.1 and 5.2). Fields in packet headers, transmitted in clear-text, will be filled with meaningful data and then padded with meaningless payloads to the appropriate lengths; here, the payloads are considered to be encrypted by attackers.

Below we describe two details. 1) By default, the ZigBee protocol does not have ACK frames. However, 802.15.4 [4] supports ACK frames as an option. If SniffMislead notices that a device indeed sends ACK frames, SniffMislead also forge them for that device. This is necessary because our fake ZigBee packets will be discarded by the destination device (as SniffMislead cannot generate valid message authentication code); thus, SniffMislead needs to generate the corresponding ACK frames to fool the attacker. Our evaluation does not find devices that send ACK frames, though. 2) If a device uses MAC randomization [67] (which is not seen in our evaluation), SniffMislead can observe this and uses the latest MAC address for forging packets. Again, as SniffMislead does not have the session key, the forged packets will be discarded by the destination device.

Finally, the created packets are injected via a packet transmitter. SniffMislead can be extended to simultaneously support multiple wireless network protocols that contain packet-level signatures (e.g., WiFi, ZigBee, and BLE). For example, to handle the ZigBee-WiFi case (a ZigBee frame is sent to some bridge node that forwards it to the cloud via WiFi), SniffMislead can be extended to inject both ZigBee and WiFi packets. That is, SniffMislead uses a ZigBee transmitter to inject a fake ZigBee packet and a WiFi transmitter to inject a fake WiFi packet.

Injected Packets in Meshed Networks. Routing/relay nodes in meshed networks only check layer 2 (e.g., MAC layer) and layer 3 (e.g., network layer) header, and they do not check the (encrypted) payloads [4]. Only the final destination node authenticates the payloads. According to the ZigBee routing mechanism, as long as we can forge a ZigBee frame with the correct MAC-layer header (including the next-hop MAC address) and network layer header (including the final destination network address), the packet can be correctly forwarded, and an attacker can not identify it.

7 EXPERIMENTAL SETUP

In this section, we present our experimental testbed (Section 7.1), selected thresholds (Section 7.2), and the attackers' methods used to evaluate the effectiveness of SniffMislead (Section 7.3).

7.1 Testbed

We configure a smart home in an apartment, using commercial IoT devices. The testbed setup is shown in Figure 5. Samsung SmartThings platform [56] is selected because it is one of the most popular and representative smart home platforms [34]. We use several ZigBee devices because many smart home IoT devices use ZigBee [31]. We employ a commercial, off-the-shelf sniffer, TelosB Dongle [60], and an open-source software tool, killerbee [48], to collect ZigBee traffic and inject decoy packets. There is one real user, with part-time roommates and irregular visitors, in our experimental environment, who is told to behave as he normally would in his home. Several common smart apps and automation rules are installed and configured. Compared to a multi-user scenario, a single-user smart home environment is more vulnerable to the proposed threat as it is easier to infer a single ongoing user behavior in a smart home [1]. Meanwhile, since both SniffMislead and existing methods [3, 33] can de-multiplex concurrent activities from multiple users, the number of real users should not affect the experimental results much.

After deployment, SniffMislead immediately starts to obtain the required information from the target smart home. Consequential wireless packets are dumped and grouped according to source and destination addresses. The dump files are then used to extract network patterns of device events, device-event-level features of behaviors, behavior associations, and the real user's daily routines.

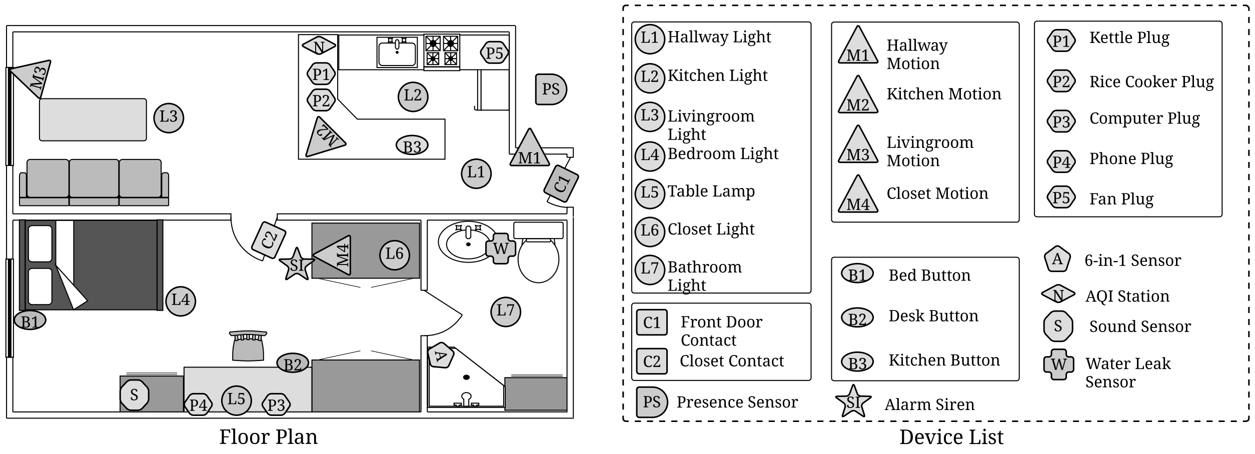

Figure 5: The experimental setup: a smart home with multiple IoT devices.

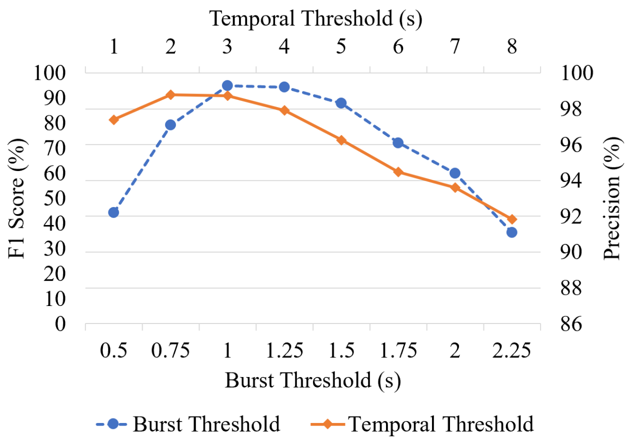

Figure 6: Evaluation results of the two thresholds. The lower x-axis shows different burst thresholds. The left y-axis shows the average F1 score of inferring device events with different burst thresholds. The upper x-axis shows different temporal thresholds. The right y-axis shows the average precision of correctly segmented event set within a behavior with different temporal thresholds.

After the first day of learning, SniffMislead is able to generate policies for placing phantom users. It continuously learns the smart home, updates its database, and improves its policies. In our testbed, the default value of λ (Section 6.1) is set to 5%, and it takes eight days to finalize the number of phantom users as eight. On average, the real user in our experimental environment generates 30 behaviors each day. To protect the user privacy, each phantom user is assigned a behavior pattern daily, with an average of 30 behaviors, 1500 simulated device events, and more than 11000 injected packets.

7.2 Threshold Selection

We need to find an appropriate burst threshold β (Section 5.2) and temporal threshold τ (Section 5.3.1), both of which may directly impact the effectiveness of SniffMislead.

Captured wireless packets, belonging to the same device, are clustered by the burst threshold β. Based on our observations, packets triggered by a device event are usually transmitted one-by-one in a very short time interval, less than one second in most cases. HoMonit [79] measures its burst threshold with integer values, ranging from 0 to 10 seconds, and finds that the best threshold is 1 second. We perform experiments on our dataset, and the burst threshold β used is from 0.5 to 2, with an interval of 0.25. We use the F1 score to indicate the accuracy of whether a device event is inferred. For each value of β, the average F1 score of all device events is calculated to denote its efficiency. As shown in Figure 6, the F1 score achieves the maximum when the threshold is 1 second. Hence, the burst threshold β is set to 1 second.

Device events belonging to temporally-distinct behaviors are clustered by the temporal threshold τ. Below are the observations from an existing work [33]: For one of their datasets, Cairo [70], only 1% of events have an inter-event time interval of 60 seconds and higher, and 93% of behaviors have inter-behavior time intervals of more than 60 seconds. For another dataset, KasterenA [66], 88% of events have an inter-event time interval less than 120 seconds, while only 22% of the behaviors have an inter-behavior time interval less than 120 seconds. Hence, we measure the temporal threshold τ from 60 to 600 seconds with an interval of 60, based on our dataset. The precision is defined as the fraction of the correctly segmented event sets within a behavior. As shown in Figure 6, the precision achieves the maximum when the threshold is 120 seconds. Hence, the temporal threshold τ is set to 120 seconds.

7.3 Playing the Role of an Attacker

To evaluate the effectiveness of SniffMislead, we play the role of an attacker and make use of possible methods to verify whether user behaviors can still be correctly inferred in the presence of phantom users and whether SniffMislead can be defeated. To infer a user behavior, attackers usually perform a two-stage attack [1], including device event inference and user behavior inference. First, based on individual packet-level signatures, attackers infer device events from encrypted wireless traffic. Next, according to the inferred device events, attackers predict user behaviors. The probability of correctly inferring behaviors relies on the effectiveness of both stages. To defeat SniffMislead, we consider a strong assumption of the attacker: he has prior knowledge of the target home, i.e., he knows the device list, the traffic pattern of each device, the home and device layout, and the configured automation rules. Based on existing works (e.g., causal relationship analysis [2, 10, 69] and context and integrity detection [26, 32, 37, 52]), we design rules to represent the prior knowledge and build a model to maintain the home context, both denoted as filters, and aimed to discard decoy packets or events that do not satisfy these rules or conform to the home context.

Device Event Inference. The attacker needs to train a detector to infer device events, hence network patterns of device events should be exacted first. Unlike the unsupervised method used by SniffMislead (Section 5.2), supervised methods are used to train a detector for device event inference, since attackers may have a supervised learning dataset in advance. Each device event is manually triggered 100 times. To extract their feature vectors, generated packets are dumped, marked, and then clustered. After obtaining feature vectors with labels from the packet sequence, a supervised learning algorithm is then applied to the dataset. One supervised learning algorithm is trained, namely Random Forest (RF) classifier, as it yields good results of device event inference [1]. The algorithm is used to detect whether injected packets by SniffMislead can be recognized as expected device events by an attacker and whether decoy packets are distinguishable from real ones.

User Behavior Inference. After device events are inferred, another detector is needed for attackers to infer user behaviors. Recent works show good performance of using Hidden Markov Model (HMM) to infer user behaviors in a smart home environment [1, 33, 45, 52]. We choose to use HMM for behavior inference. The characteristics of smart homes make HMM quite suitable to be applied in a smart home environment [1, 53]. User behaviors are considered as hidden states. User behaviors cause changes to the state of IoT devices, defined as visible states. Each hidden state generates one of the defined visible states. Any user behavior in a smart home can be predicted by observing the states of the IoT devices. An attacker's goal would be to infer the hidden state (i.e., user behaviors) from the visible state (i.e., states of IoT devices) using HMM. States of IoT devices can be illustrated with binary output, "1" for the active status and "0" for inactive status. For a specific time, the smart home state can be represented as an n-bit binary number with $m = 2^n$ possible states, where n is the total number of device states in the smart home. The most likely sequence of hidden states can be found by observing smart devices, using the Viterbi algorithm [68]. The proposed HMM detector for user behavior inference is trained using real smart home traffic, without SniffMislead. This detector is then used to test the effectiveness of inferring user behaviors, after SniffMislead is used.

Table 1: Results of device event inference

		Real events			Decoy events		
Device	Event	Recall(%)	Precision(%)	F1(%)	Recall(%)	Precision(%)	F1(%)
Contact Sensor	open	97	98	97.5	97	98	97.5
Motion Sensor	active	98	98	98	98	98	98
Plug	turn on	100	99	99.5	100	100	100
Button	click	100	100	100	100	100	100
Presence Sensor	present	94	93	93.5	92	94	93
Alarm Siren	alarm	93	91	92	93	91	92
Water Leak Sensor	wet	95	98	96.5	96	97	96.5

Table 2: Results of user behavior inference

Precision (%) \ Num / Behavior	0	1	2	3	4	5	6	7	8
Leave home	99	54	31	26	23	17	10	8	6
Return home	98	54	34	28	26	18	11	7	5
Fall asleep	95	51	32	25	24	20	11	6	4
Wake up	96	53	30	25	22	18	11	6	3
Cooking	91	45	27	16	12	11	8	4	2
Toilet use	92	43	29	28	15	9	9	4	2
Bathing	94	48	30	20	17	10	8	6	4
Getting dressed	95	39	30	22	14	8	7	5	3
Working	94	46	29	20	16	10	8	5	3

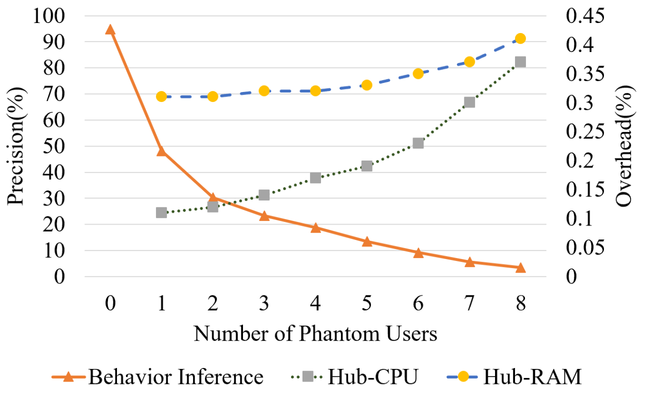

Figure 7: Evaluation results of SniffMislead with different number of phantom users. The x-axis shows number of phantom users. The left y-axis shows the average accuracy of correctly inferring real user behaviors. The right y-axis shows the increase of CPU and RAM usage, compared with the case when there is no phantom user.

8 EVALUATION RESULTS

We present the evaluation results in this section, including the effectiveness of privacy protection (Section 8.1) and the overhead and influence to smart home (Section 8.2).

8.1 Efficiency of Privacy Protection

In our experiment, SniffMislead is used to simulate phantom users, starting from one user incrementally. We play the role of an attacker, using the two detectors to infer device events and user behaviors, and the filters to discard injected decoy packets and events (both discussed in Section 7.3).

Table 1 shows the evaluation results of inferring some device events. The recall of the device event inference is the ratio of successful inferred events to all triggered events. The precision of device event inference is the ratio of correctly-inferred events to all inferred events. The F1 score is simply the harmonic mean of precision and recall. There are no noticeable differences in the recall, precision, or F1 score, between real and decoy packets, meaning that an attacker would recognize the injected packets as expected device events of real users. These decoy events would "change" device states, from the attacker's perspective, causing the attacker to infer wrong device states. However, it needs to be emphasized that SniffMislead can not always prevent attackers from inferring correct device states as a decoy event sometimes lets the device be "in" its real state (e.g., three successive events: (real) `ON` $\rightarrow$ (decoy) `OFF` $\rightarrow$ (decoy) `ON`). We measure the percentage of cases when the attacker knows a device's state correctly with the existence of SniffMislead. The percentage tends to be $\frac{1}{k}$, and k is the number of mutually-exclusive states of a smart device, during the long-run period. For instance, a light has two mutually-exclusive states (i.e., `ON` and `OFF`), and the attacker can only infer its state for 50% of the time, which is not better than a blind guess.

Table 2 shows the evaluation results of inferring nine typical user behaviors, with different numbers of phantom users. The precision of behavior inference is the ratio of correctly-inferred real user behaviors to all behaviors of the same type caused by real users. The average precision is calculated to indicate the effectiveness of SniffMislead simulating different number of phantom users. As shown in Figure 7, the average precision of the behavior inference decreases as the number of phantom users increases. We find that using eight phantom users is very effective in our testbed, undermining a wireless sniffing attacker's capability of behavior inferences from 94.8% to 3.5%, also less than λ. When attackers have lots of behavior inferring errors, user privacy is protected. In our experimental environment, on average, each day, the real user generates about 30 behaviors while our system approximately injects 240 fake behaviors (in the case of 8 phantom users), and only one real-user behavior is correctly inferred.

We design SniffMislead as a much more advanced solution than random event injection, which is an intuitive solution that uses random noise to hide real events. We perform micro-benchmark experiments to evaluate random event injection against an attacker. We modify killerbee [48] to randomly generate device events for the target smart home and use TelosB Dongle [60] to inject corresponding packets. Without the filters, random event injection seemed to have some effect. The average accuracy of inferring real user behaviors decreases by 52% because methods like HMM detectors only care about the current states of devices, and randomly injected events can change device states to some extent. However, they are still less efficient than SniffMislead. With the existence of the filters, the efficiency of random event injection becomes even worse. Results show that around 76% of random events are discarded. After that, we use the HMM detector and obtain an average accuracy of 93%, implying that random event injection could hardly protect user privacy. This is because many randomly injected decoy events could not change the home context (i.e., they cannot "change" what a real user is doing) or they suffer from logical errors. On the contrary, only less than 0.05% decoy events injected by SniffMislead are discarded, i.e., SniffMislead still works well facing such filters.

To sum up, SniffMislead is a much more robust solution than straightforward and intuitive methods (e.g., random event injection). In our experiment, SniffMislead can defend against attackers who use sniffers to analyze packet-level signatures of wireless packets and infer device states and user behaviors. SniffMislead can protect user privacy at both device-state and user-behavior levels.

8.2 Overhead and Influence to Smart Home

Injected packets may cause network latency or performance overhead to the smart home infrastructure, because those packets may need to be filtered out by smart devices. Since the SmartThings Hub is close-sourced, we can not measure the overhead directly. Instead, we use a Raspberry Pi 3B and deploy it with OpenHub [58], to simulate the network functions of SmartThings Hub, as injected decoy packets are filtered at the network level. The CPU and RAM usage of the Raspberry Pi 3B are recorded every second, with and without SniffMislead, with different numbers of phantom users. The average usage of CPU and RAM is calculated to show the overhead. Figure 7 gives the evaluation results. The CPU overhead ranges from 0.1% to 0.4%, and the RAM overhead ranges from 0.3% to 0.45%. Both overheads are small and acceptable. With more phantom users simulated, the overhead increases, which is reasonable.

SniffMislead has a very small impact on the power consumption of battery-operated devices. First, ZigBee end devices that are powered by batteries are not designed to be awakened by RF signals. Instead, they use periodic polling and stay in the sleep mode otherwise [47]. It means that our injected packets do not wake up the device and hence do not increase the RF power consumption. Second, we have a detailed experiment on a Raspberry Pi 3B as for the computing power consumption. The injected packets only increase the CPU usage by 0.4%. Lastly, SniffMislead only injects packets when needed. Compared with other defenses, such as continuously injecting a lot of packets randomly, SniffMislead has its strength.

SniffMislead achieves its goals by injecting decoy packets. People may worry if those injected packets could have side effects, such as accidental device state changes or network latency to their smart home. Actually, SniffMislead causes no security violation. IoT protocols are usually designed to drop invalid packets [24, 30, 63] (e.g., to prevent a replay attack or message spoofing). Using SniffMislead, payloads of injected packets are meaningless. Also, SniffMislead only injects packets when needed, avoiding unnecessary packet injections. To detect if there are possible device state errors caused by SniffMislead, we develop a smart app and install it in the SmartThings platform in our testbed, which is used to monitor smart devices' state changes. Each time packets of a decoy event are injected, the current states of devices are checked to detect any unexpected state change. Our experimental results show that the injected packets cause no device state errors. The timestamps of device events available in the SmartThings console are used to calculate the latency of events. In two settings: whether SniffMislead is deployed or not, the average latency of each event shows no significant difference. Moreover, according to the feedback from the user involved in our experiment, no noticeable delay was experienced. All the above shows that SniffMislead runs in a secure and non-intrusive manner, and causes no side effects to the target smart home and the users.

9 DISCUSSION

9.1 Generality and Applicability

In our current work, we evaluate SniffMislead in the SmartThings platform with ZigBee devices. All the functions of SniffMislead are only based on the network traffic patterns of the target smart home, having nothing to do with specific platforms or protocols. With some modifications, SniffMislead can be applied to other smart home platforms and devices using other network protocols. Most IoT devices in a smart home environment are power-constrained, and the employed wireless protocols are lightweight (e.g., Z-Wave, ZigBee, BLE, and WiFi). These lightweight protocols, which are typically designed for low transmission rate and reduced data redundancy for low power consumption, containing packet-level signatures.

In this work, we mainly consider wireless packet sniffers. We do not consider other types of attackers, such as Wide Area Network (WAN) sniffers [64], which target outbound traffic between a home router and an Internet Service Provider (ISP) network. However, SniffMislead can be modified to defend against WAN sniffers. The idea of injecting packets into a wired network to simulate phantom users would still work, e.g., the home router can be modified to run the function of SniffMislead.

Side-channel information leakage of encrypted wireless network traffic has its beneficial use; for example, it enables detection of misbehaving smart apps [79]. A concern is that SniffMislead's injected packets may decrease this kind of security tool's detection accuracy. One solution is that SniffMislead gets integrated with these security tools, which thus knows which packets are injected by SniffMislead.

9.2 Limitations

Real users may cause conflicts in device statuses, which cannot be avoided by SniffMislead as SniffMislead cannot predict the behaviors of real users. For example, SniffMislead creates a fake "Window-Opened" event at 10:00, and then a real user opens the window, creating a "Window-Opened" event at 10:15 again. An attacker who observes the two events may know that the event at 10:00 is fake and that the one at 10:15 is real because SniffMislead does not create a "Window-Opened" event for a window that is already open. To deal with this, our future work plans to extend SniffMislead as follows. First, it could inject packets after the conflicts arise to mislead the attacker's inference about the current statuses of IoT devices. For example, if a phantom user closes the window soon (with a random duration) after 10:15, then the attacker is not sure about the current status of the window. Second, SniffMislead could occasionally create a small number of conflicts, so the attacker is not sure whether a conflict has been caused by the real user or SniffMislead. For example, it can sometimes inject a fake "Window-Opened" event after a real "Window-Opened" event.

SniffMislead cannot inject decoy packets for IoT devices that use cellular networks (e.g., 4G/5G) over a licensed spectrum, because SniffMislead cannot use a licensed spectrum. Note that most existing IoT devices do not use cellular networks, which incur monthly fees.

Another potential limitation of SniffMislead is that an attacker may try to identify packets injected by SniffMislead using wireless channel characteristics, e.g., received signal strength indicator (RSSI) and channel-state information (CSI).

The literature (e.g., [78]) shows localization based on RSSI does not perform well when the receiver is placed in a room different from that contains IoT devices and SniffMislead. Moreover, RSSI-based localization is considered inaccurate, particularly in indoor environments [76]. We can defeat RSSI-based localization by placing a SniffMislead instance in each room. An instance only injects packets for devices in the same room (i.e., presumably devices with strong RSSI), so the attacker cannot distinguish a SniffMislead instance from other IoT devices. A detailed study of this aspect will be our future work.

An attacker who has control of a device for collecting CSI is able to perform more precise localization and thus can distinguish between packets from SniffMislead and those from real IoT devices. The threat model of this work (Section 3.1) is mainly concerned with a remote attacker who sniffs packets based on a compromised IoT device at the target home. Note that collecting CSI requires a special network interface card (NIC) [75], and most IoT devices do not have this kind of NIC.

A local attacker who can *physically* place a device with a special NIC very close the target home can collect CSI, and in this case SniffMislead will fail, which we admit as a limitation. To successfully launch this attack, the attacker needs to be physically very close to the target home and place a device near the home. The attacker takes risks because he may be captured by the home's security camera or doorbell camera.

On the other hand, SniffMislead still has its values, as it can defeat most remote attackers, who can harvest privacy-sensitive information of a large number of smart homes at scale.

10 CONCLUSION

We proposed SniffMislead, which can effectively defend against passive attackers who sniff encrypted wireless traffic to infer user privacy-sensitive information. SniffMislead is a non-intrusive wireless packet injection tool, developed using a stand-alone device, without modifying IoT devices, hubs, platforms, or communication protocols. It works without requiring manual configurations. SniffMislead uses a top-down approach to place indistinguishable *phantom users*, who "live" in a home like real users would, preventing attackers from making reliable inferences about device states and user behaviors. The real smart home testbed evaluation showed that SniffMislead significantly reduced a wireless sniffing attacker's capability of behavior inference; as a result, the attackers could not be able to tell whether an inferred event was real or not and could not distinguish between real and phantom users. Therefore, SniffMislead can provide effective, resilient and self-adaptive privacy protection for smart homes.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers and our shepherd Dr. Ramin Sadre for their invaluable suggestions. This material is partially based upon work supported by the National Science Foundation under Grants CNS-1856380, CNS-2016415, CNS-2107093, CNS-1828363, CNS-1564128, CNS-1824440, and CNS-2016589. The first author was supported by China Scholarship Council.

REFERENCES

[1] 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!. In *ACM Conference on Security and Privacy in Wireless and Mobile Networks.*

[2] Mehmet S Aktas and Merve Astekin. 2019. Provenance aware run-time verification of things for self-healing Internet of Things applications. *Concurrency and Computation: Practice and Experience* 31, 3 (2019), e4263.

[3] Mohammad Arif Ul Alam, Nirmalya Roy, and Archan Misra. 2019. Tracking and Behavior Augmented Activity Recognition for Multiple Inhabitants. *IEEE Transactions on Mobile Computing* (2019).

[4] The ZigBee Alliance. 2015. ZigBee Specification. https://zigbeealliance.org/wp-content/uploads/2019/11/docs-05-3474-21-0csg-zigbee-specification.pdf.

[5] Omar Alrawi, Chaz Lever, Manos Antonakakis, and Fabian Monrose. 2019. Sok: Security evaluation of home-based iot deployments. In *IEEE symposium on security and privacy.*

[6] 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* 2019, 3 (2019), 128–148.

[7] Noah Apthorpe, Dillon Reisman, and Nick Feamster. 2017. Closing the blinds: Four strategies for protecting smart home privacy from network observers. *arXiv preprint arXiv:1705.06809* (2017).

[8] Noah Apthorpe, Dillon Reisman, and Nick Feamster. 2017. A smart home is no castle: Privacy vulnerabilities of encrypted iot traffic. *arXiv preprint arXiv:1705.06805* (2017).

[9] 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).

[10] Leonardo Babun, Amit Kumar Sikder, Abbas Acar, and A Selcuk Uluagac. 2018. Iotdots: A digital forensics framework for smart environments. *arXiv preprint arXiv:1809.00745* (2018).

[11] Iulia Bastys, Musard Balliu, and Andrei Sabelfeld. 2018. If this then what? Controlling flows in IoT apps. In *ACM SIGSAC conference on computer and communications security.*

[12] Z Berkay Celik, Leonardo Babun, Amit Kumar Sikder, Hidayet Aksu, Gang Tan, Patrick McDaniel, and A Selcuk Uluagac. 2018. Sensitive information tracking in commodity IoT. In *USENIX Security Symposium.*

[13] Z Berkay Celik, Gang Tan, and Patrick D McDaniel. 2019. IoTGuard: Dynamic Enforcement of Security and Safety Policy in Commodity IoT.. In *Network and Distributed Systems Security Symposium.*

[14] Yunang Chen, Amrita Roy Chowdhury, Ruizhe Wang, Andrei Sabelfeld, Rahul Chatterjee, and Earlence Fernandes. 2021. Data Privacy in Trigger-Action Systems. In *IEEE Symposium on Security and Privacy.*

[15] Haotian Chi, Qiang Zeng, Xiaojiang Du, and Lannan Luo. 2021. PFirewall: Semantics-Aware Customizable Data Flow Control for Smart Home Privacy Protection. In *Network and Distributed Systems Security Symposium.*

[16] Haotian Chi, Qiang Zeng, Xiaojiang Du, and Jiaping Yu. 2020. Cross-app interference threats in smart homes: Categorization, detection and handling. In *IEEE/IFIP International Conference on Dependable Systems and Networks.*

[17] Jiwon Choi, Hayoung Jeoung, Jihun Kim, Youngjoo Ko, Wonup Jung, Hanjun Kim, and Jong Kim. 2018. Detecting and identifying faulty IoT devices in smart home with context extraction. In *IEEE/IFIP International Conference on Dependable Systems and Networks.*

[18] Sungjoon Choi, Eunwoo Kim, and Songhwai Oh. 2013. Human behavior prediction for smart homes using deep learning. In *International Symposium on Robot and Human Interactive Communication.*

[19] Bogdan Copos, Karl Levitt, Matt Bishop, and Jeff Rowe. 2016. Is anybody home? inferring activity from smart home network traffic. In *IEEE Security and Privacy Workshops.*

[20] Trisha Datta, Noah Apthorpe, and Nick Feamster. 2018. A developer-friendly library for smart home IoT privacy-preserving traffic obfuscation. In *Workshop on IoT Security and Privacy.*

[21] Tamara Denning, Tadayoshi Kohno, and Henry M Levy. 2013. Computer security and the modern home. *Commun. ACM* 56, 1 (2013), 94–103.

[22] Wenbo Ding and Hongxin Hu. 2018. On the safety of iot device physical interaction control. In *ACM SIGSAC Conference on Computer and Communications Security.*

[23] 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. In *IEEE symposium on security and privacy.*

[24] Xueqi Fan, Fransisca Susan, William Long, and Shangyan Li. 2017. Security analysis of zigbee. *MWR InfoSecurity* (2017), 1–18.

[25] Earlence Fernandes, Justin Paupore, Amir Rahmati, Daniel Simionato, M. Conti, and Atul Prakash. 2016. FlowFence: Practical Data Protection for Emerging IoT Application Frameworks. In *USENIX Security Symposium.*

[26] Earlence Fernandes, Amir Rahmati, Jaeyeon Jung, and Atul Prakash. 2018. Decentralized action integrity for trigger-action IoT platforms. In *Network and Distributed System Security Symposium.*

[27] Anthony Fleury, Michel Vacher, and Norbert Noury. 2009. SVM-based multimodal classification of activities of daily living in health smart homes: sensors, algorithms, and first experimental results. *IEEE transactions on information technology in biomedicine* 14, 2 (2009), 274–283.

[28] Chenglong Fu, Qiang Zeng, and Xiaojiang Du. 2021. Hawatcher: Semantics-aware anomaly detection for appified smart homes. In *USENIX Security Symposium.*

[29] Jun Han, Albert Jin Chung, Manal Kumar Sinha, Madhumitha Harishankar, Shijia Pan, Hae Young Noh, Pei Zhang, and Patrick Tague. 2018. Do you feel what I hear? Enabling autonomous IoT device pairing using different sensor types. In *IEEE Symposium on Security and Privacy.*

[30] HKCERT. 2020. Device (ZigBee) Security Study. https://www.hkcert.org/f/blog/264453/3a1c8eed-012c-4b59-9d9e-971001d66c77-DLFE-14602.pdf.

[31] Mordor Intelligence. 2020. ZigBee Market - growth, trends, and forecast (2020 - 2025). https://www.mordorintelligence.com/industry-reports/zigbee-market.

[32] Yunhan Jack Jia, Qi Alfred Chen, Shiqi Wang, Amir Rahmati, Earlence Fernandes, Zhuoqing Morley Mao, Atul Prakash, and SJ Unviersity. 2017. ContexIoT: Towards Providing Contextual Integrity to Appified IoT Platforms.. In *Network and Distributed Systems Security Symposium.*

[33] Palanivel Kodeswaran, Ravindranath Kokku, Madhumita Mallick, and Sayandeep Sen. 2016. Demultiplexing activities of daily living in IoT enabled smarthomes. In *IEEE International Conference on Computer Communications.*

[34] Joo Kyung-don. 2020. Samsung's SmartThings app has gathered 112 million subscribers around the world, with 52 million of them recognized as active users. https://en.yna.co.kr/view/AEN20200108006700320.

[35] Sanghak Lee, Jiwon Choi, Jihun Kim, Beumjin Cho, Sangho Lee, Hanjun Kim, and Jong Kim. 2017. FACT: Functionality-centric access control system for IoT programming frameworks. In *ACM on Symposium on Access Control Models and Technologies.*

[36] Manuel Lopez-Martin, Belen Carro, Antonio Sanchez-Esguevillas, and Jaime Lloret. 2017. Network traffic classifier with convolutional and recurrent neural networks for Internet of Things. *IEEE Access* 5 (2017), 18042–18050.

[37] Altti Ilari Maarala, Xiang Su, and Jukka Riekki. 2016. Semantic reasoning for context-aware Internet of Things applications. *IEEE Internet of Things Journal* 4, 2 (2016), 461–473.

[38] Yair Meidan, Michael Bohadana, Asaf Shabtai, Juan David Guarnizo, Martín Ochoa, Nils Ole Tippenhauer, and Yuval Elovici. 2017. ProfilIoT: a machine learning approach for IoT device identification based on network traffic analysis. In *Symposium on Applied Computing.*

[39] Weizhi Meng, Wenjuan Li, Chunhua Su, Jianying Zhou, and Rongxing Lu. 2017. Enhancing trust management for wireless intrusion detection via traffic sampling in the era of big data. *Ieee Access* 6 (2017), 7234–7243.

[40] Nour Moustafa, Benjamin Turnbull, and Kim-Kwang Raymond Choo. 2018. An ensemble intrusion detection technique based on proposed statistical flow features for protecting network traffic of internet of things. *IEEE Internet of Things Journal* 6, 3 (2018), 4815–4830.

[41] Qin Ni, Ana Belén García Hernando, and Iván Pau de la Cruz. 2016. A context-aware system infrastructure for monitoring activities of daily living in smart home. *Journal of Sensors* 2016 (2016).

[42] TJ OConnor, Reham Mohamed, Markus Miettinen, William Enck, Bradley Reaves, and Ahmad-Reza Sadeghi. 2019. HomeSnitch: behavior transparency and control for smart home IoT devices. In *Conference on Security and Privacy in Wireless and Mobile Networks.*

[43] Mikko Ohtamaa. 2017. Levenshtein Python C Extension Module. https://github.com/miohtama/python-Levenshtein.

[44] Antônio J Pinheiro, Jeandro M Bezerra, and Divanilson R Campelo. 2018. Packet Padding for Improving Privacy in Consumer IoT. In *IEEE Symposium on Computers and Communications.*

[45] Parisa Rashidi, Diane J Cook, Lawrence B Holder, and Maureen Schmitter-Edgecombe. 2010. Discovering activities to recognize and track in a smart environment. *IEEE transactions on knowledge and data engineering* 23, 4 (2010), 527–539.

[46] Jingjing Ren, Daniel J Dubois, David Choffnes, Anna Maria Mandalari, Roman Kolcun, and Hamed Haddadi. 2019. Information exposure from consumer iot devices: A multidimensional, network-informed measurement approach. In *Internet Measurement Conference.*

[47] ScienceDirect. 2008. Zigbee End Device. https://www.sciencedirect.com/topics/computer-science/zigbee-end-device.

[48] River Loop Security. 2021. Killerbee. https://github.com/riverloopsec/killerbee.

[49] Hardik Shah. 2021. Introduction to BLE security for IoT. https://www.simform.com/iot-bluetooth-security-vulnerabilities/.

[50] Mustafizur R Shahid, Gregory Blanc, Zonghua Zhang, and Hervé Debar. 2018. IoT devices recognition through network traffic analysis. In *IEEE International Conference on Big Data*.

[51] Sharon Shea. 2018. What is Z-Wave? https://internetofthingsagenda.techtarget.com/definition/Z-Wave.

[52] Amit Kumar Sikder, Hidayet Aksu, and A Selcuk Uluagac. 2017. 6thsense: A context-aware sensor-based attack detector for smart devices. In *USENIX Security Symposium*.

[53] Amit Kumar Sikder, Giuseppe Petracca, Hidayet Aksu, Trent Jaeger, and A Selcuk Uluagac. 2018. A survey on sensor-based threats to internet-of-things (iot) devices and applications. *arXiv preprint arXiv:1802.02041* (2018).

[54] Arunan Sivanathan, Hassan Habibi Gharakheili, Franco Loi, Adam Radford, Chamith Wijenayake, Arun Vishwanath, and Vijay Sivaraman. 2018. Classifying IoT devices in smart environments using network traffic characteristics. *IEEE Transactions on Mobile Computing* 18, 8 (2018), 1745–1759.

[55] Arunan Sivanathan, Daniel Sherratt, Hassan Habibi Gharakheili, Adam Radford, Chamith Wijenayake, Arun Vishwanath, and Vijay Sivaraman. 2017. Characterizing and classifying IoT traffic in smart cities and campuses. In *IEEE Conference on Computer Communications Workshops*.

[56] SmartThings. 2020. Samsung SmartThings. https://www.smartthings.com.

[57] Vijay Srinivasan, John Stankovic, and Kamin Whitehouse. 2008. Protecting your daily in-home activity information from a wireless snooping attack. In *International Conference on Ubiquitous Computing*.

[58] SYNOPSYS. 2020. OpenHub. https://www.openhub.net.

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

[60] Crossbow Technology. 2020. TelosB. https://www.willow.co.uk/TelosB_Datasheet.pdf.

[61] Yuan Tian, Nan Zhang, Yueh-Hsun Lin, XiaoFeng Wang, Blase Ur, Xianzheng Guo, and Patrick Tague. 2017. Smartauth: User-centered authorization for the internet of things. In *USENIX Security Symposium*.

[62] Robert Tibshirani, Guenther Walther, and Trevor Hastie. 2001. Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 63, 2 (2001), 411–423.

[63] Jonathan Tournier, François Lesueur, Frédéric Le Mouël, Laurent Guyon, and Hicham Ben-Hassine. 2020. A survey of IoT protocols and their security issues through the lens of a generic IoT stack. *Internet of Things* (2020), 100264.

[64] Rahmadi Trimananda, Janus Varmarken, Athina Markopoulou, and Brian Demsky. 2020. Packet-level signatures for smart home devices. In *Network and Distributed Systems Security Symposium*, Vol. 2020.

[65] Mostafa Uddin, Tamer Nadeem, and Santosh Nukavarapu. 2019. Extreme SDN Framework for IoT and Mobile Applications Flexible Privacy at the Edge. In *IEEE International Conference on Pervasive Computing and Communications*.

[66] TLM Van Kasteren, Gwenn Englebienne, and Ben JA Kröse. 2010. Activity recognition using semi-Markov models on real world smart home datasets. *Journal of ambient intelligence and smart environments* 2, 3 (2010), 311–325.

[67] Mathy Vanhoef, Célestin Matte, Mathieu Cunche, Leonardo S Cardoso, and Frank Piessens. 2016. Why MAC address randomization is not enough: An analysis of Wi-Fi network discovery mechanisms. In *ACM on Asia conference on computer and communications security*.

[68] Andrew Viterbi. 1967. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE transactions on Information Theory* 13, 2 (1967), 260–269.

[69] Qi Wang, Wajih Ul Hassan, Adam Bates, and Carl Gunter. 2018. Fear and logging in the internet of things. In *Network and Distributed System Security Symposium*.

[70] WSU. 2009. Wsu casas dataset. http://ailab.wsu.edu/casas/datasets/.

[71] Sijie Xiong, Anand D Sarwate, and Narayan B Mandayam. 2018. Defending against packet-size side-channel attacks in IoT networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing*.

[72] Rixin Xu, Qiang Zeng, Liehuang Zhu, Haotian Chi, and Xiaojiang Du. 2018. Privacy Leakage in Smart Homes and Its Mitigation: IFTTT as a Case Study. In *International Performance Computing and Communications Conference*.

[73] Abdulsalam Yassine, Shailendra Singh, and Atif Alamri. 2017. Mining human activity patterns from smart home big data for health care applications. *IEEE Access* 5 (2017), 13131–13141.

[74] Li Yujian and Liu Bo. 2007. A normalized Levenshtein distance metric. *IEEE transactions on pattern analysis and machine intelligence* 29, 6 (2007), 1091–1095.

[75] Faheem Zafari, Athanasios Gkelias, and Kin K Leung. 2019. A survey of indoor localization systems and technologies. *IEEE Communications Surveys & Tutorials* 21, 3 (2019), 2568–2599.

[76] Andrea Zanella. 2016. Best practice in RSS measurements and ranging. *IEEE Communications Surveys & Tutorials* 18, 4 (2016), 2662–2686.

[77] Qiang Zeng, Mingyi Zhao, Peng Liu, Poonam Yadav, Seraphin Calo, and Jorge Lobo. 2014. Enforcement of autonomous authorizations in collaborative distributed query evaluation. *IEEE Transactions on Knowledge and Data Engineering* 27, 4 (2014), 979–992.

[78] Jiansong Zhang, Zeyu Wang, Zhice Yang, and Qian Zhang. 2017. Proximity based IoT device authentication. In *IEEE International Conference on Computer Communications*.

[79] Wei Zhang, Yan Meng, Yugeng Liu, Xiaokuan Zhang, Yinqian Zhang, and Haojin Zhu. 2018. Homonit: Monitoring smart home apps from encrypted traffic. In *ACM SIGSAC Conference on Computer and Communications Security*.