Evasion Attacks and Defenses on Smart Home Physical Event Verification

Muslum Ozgur Ozmen, Ruoyu Song, Habiba Farrukh, and Z. Berkay Celik
Purdue University
{mozmen, song464, hfarrukh, zcelik}@purdue.edu

***Abstract*—In smart homes, when an actuator's state changes, it sends an event notification to the IoT hub to report this change (e.g., the door is unlocked). Prior works have shown that event notifications are vulnerable to spoofing and masking attacks. In event spoofing, an adversary reports to the IoT hub a fake event notification that did not physically occur. In event masking, an adversary suppresses the notification of an event that physically occurred. These attacks create inconsistencies between physical and cyber states of actuators, enabling an adversary to indirectly gain control over safety-critical devices by triggering IoT apps. To mitigate these attacks, event verification systems (EVS), or broadly IoT anomaly detection systems, leverage physical event fingerprints that describe the relations between events and their influence on sensor readings. However, smart homes have complex physical interactions between events and sensors that characterize the event fingerprints. Our study of the recent EVS, unfortunately, has revealed that they widely ignore such interactions, which enables an adversary to evade these systems and launch successful event spoofing and masking attacks without getting detected.**

In this paper, we first explore the evadable physical event fingerprints and show that an adversary can realize them to bypass the EVS given the same threat model. We develop two defenses, EVS software patching and sensor placement, with the interplay of physical modeling and formal analysis, to generate robust physical event fingerprints and demonstrate how they can be integrated into the EVS. We evaluate the effectiveness of our approach in two smart home settings that contain 12 actuators and 16 sensors when two different state-of-the-art EVS are deployed. Our experiments demonstrate that 71% of their physical fingerprints are vulnerable to evasion. By incorporating our approach, they build robust physical event fingerprints, and thus, properly mitigate realistic attack vectors.

I. Introduction

Smart homes include various sensors and actuators. Sensors continuously measure physical channels (e.g., sound) and output numerical or boolean-typed readings. Actuators influence physical channels by executing actuation commands. When an actuation command is invoked, the actuator changes its state and sends an event notification to the IoT hub to report this change [17], [38]. For instance, when the user opens the window, the window reports the `window-open` event.

Event notifications are essential for the seamless operation of smart homes for two reasons. First, the IoT hub monitors the actuator states through these notifications and notifies the user of any important change (e.g., when the fire alarm sounds). Second, the IoT hub activates other actuators based on the IoT apps installed in the smart home, e.g., unlocks the door and turns on the lights when the `alarm-on` notification is received [18].

Prior works have shown that event notifications are vulnerable to spoofing and masking attacks [31], [84], [87]. Event spoofing occurs when an adversary reports a fake event notification that did not physically occur, e.g., the adversary spoofs an `alarm-on` event where the alarm's physical state is off. Event masking occurs when an adversary prevents an actuator from sending the notification of an event that physically occurred, e.g., the adversary intercepts the `window-open` event notification when the window physically opens.

Existing works have shown that an adversary can conduct event spoofing and masking attacks in different ways. One line of work exploits phantom devices, which are computer programs that mimic a real physical device in a smart home, to conduct these attacks [83], [87]. Another line of work shows the adversary can use malicious IoT apps due to the design flaws in IoT programming frameworks [31], [75], [84]. These attacks are stealthy as they do not require the physical control of devices, but they allow an adversary to perform remote event spoofing and masking attacks.

Event spoofing and masking attacks enable an adversary to create discrepancies between actuators' cyber and physical states. These discrepancies allow an adversary to stealthily gain control over sensitive devices through IoT apps. For instance, an adversary can unlock the patio door by spoofing a `light-on` event if an app opens the patio door when the lights turn on.

There is a growing interest in event verification systems (EVS) to protect smart homes from event spoofing and masking attacks through *physical event fingerprints* [11], [12], [34], [67], [68], [81]. EVS extract physical event fingerprints that define the relations between events and sensor readings to flag inconsistencies between cyber and physical actuator states at run-time. However, complex physical relations exist between events and sensor readings [19], [27], [56]: (1) An event influences single or multiple physical channels, (2) a sensor measures the influence from a single event or the aggregated influence from multiple events, and (3) the sensor readings depend on the distance between an actuator and sensor where they decrease with an increasing distance. To build accurate event fingerprints, such physical relations must be incorporated into EVS. Unfortunately, our study of state-of-the-art EVS has shown they broadly ignore this requirement. We show that these relations enable evasion attacks against EVS, where an adversary launches successful spoofing and masking attacks.

Network and Distributed System Security (NDSS) Symposium 2023
27 February - 3 March 2023, San Diego, CA, USA
ISBN 1-891562-83-5
https://dx.doi.org/10.14722/ndss.2023.23070
www.ndss-symposium.org

In this paper, we propose
against evasion attacks. Given
generated by an EVS, we first
to evasion by analyzing the ph
and sensor readings. For each
we patch EVS software by intr
which define the relation betv
joint influence on sensor rea
software patching is limited ir
events influence sensor readi
we introduce sensor location
approach, which finds sensor
events have unique fingerprint
software patching and sensor lo
fingerprints that can detect eva

We evaluate our defense in
studio apartment with 7 actuat
room with 5 actuators and 7
of-the-art EVS (HAWatcher [
settings. We identify 75% of th
by HAWatcher and 67% of
vulnerable to evasion. We show that software patching prevents 64% of the evasion attacks against HAWatcher, and 40% of the attacks against Peeves. We then conduct location patching by relocating two sensors in the studio apartment and two sensors in the patient room. The fingerprints derived after sensor location patching prevent all remaining evasion attacks, ensuring an adversary cannot evade HAWatcher and Peeves.

In this work, we make the following contributions:

- We propose an algorithm that discovers the evadable physical event fingerprints by analyzing if they can be satisfied or concealed by the influences from other events.
- We develop two defenses against evasion attacks, EVS software patching and sensor location patching. Software patching is an automated method that introduces new physical fingerprints into EVS. Sensor location patching is a security-by-design approach complementary to software patching, which generates a sensor placement that ensures physical fingerprints are unique to an event.
- We evaluate our defenses with two separate EVS deployed in two smart homes with a total of 12 actuators and 16 sensors. Our evaluation demonstrates 71% of the physical fingerprints are susceptible to evasion attacks, and our system can successfully prevent all.
- Our system is available at
  https://github.com/purseclab/EVS_Evasion
  for public use and validation.

II. Background

A. Event Spoofing and Masking Attacks

Smart homes rely on an IoT hub that monitors and controls the IoT sensors and actuators. Sensors continuously measure physical channels and output numerical or boolean-typed readings, e.g., a sound sensor measures ambient sound. Actuators influence physical channels by executing an actuation command, e.g., `heater-on` changes the temperature. We refer to *events* as actuator state changes that are triggered through IoT apps, direct physical interaction of users with actuators, or

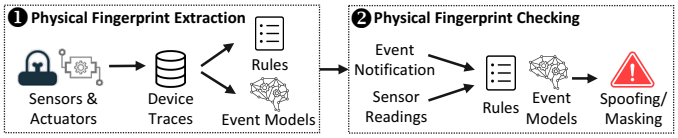


Fig. 1: Overview of event verification systems (EVS)

user interfaces such as companion apps and virtual assistants. For instance, a user can manually lock a smart door, can issue a "lock my door" voice command, and can use an IoT app that locks the door when they leave home. When a physical event occurs, the actuator sends an *event notification* to the hub to notify its state change, e.g., the door sends "door is locked" notification to the hub after executing the command. The hub updates the actuator's *cyber state* upon receiving the notification. The physical events are the phenomena that happen in the smart home, and the event notification represents how the IoT hub monitors the cyber state of that physical phenomena.

Prior works have shown that smart homes are vulnerable to event spoofing and masking attacks [31], [84], [87] (Detailed in Section III). *Event spoofing* occurs when an adversary reports a fake event that has not physically occurred. For instance, the adversary spoofs the light-on event, although the light's physical state is off. By spoofing light-on, the adversary can physically unlock the patio door if there is an IoT app that unlocks the patio door when the light turns on. *Event masking* occurs when the adversary suppresses the notification of a physical event. In this way, they prevent the IoT hub from receiving event notifications and impede invoking device actuations. For instance, they mask the alarm-on notification when there is a fire, and block an app that unlocks the door for fire safety.

B. Event Verification Systems

To protect smart homes against event spoofing and masking, there is an increasing interest in event verification systems (EVS) (and more broadly, IoT anomaly detection systems) from both the research community [11], [12], [13], [22], [34], [64], [67], [68], [25] and industry [39], [63], [70]. These systems use *physical event fingerprints* between events and their influence on sensor readings to verify event notifications. Broadly, we group EVS into two types based on how they learn event fingerprints, rule-based (R-EVS), and ML-based (ML-EVS). Figure 1 shows their two common stages: (❶) physical fingerprint extraction and (❷) physical fingerprint checking.

Physical Fingerprint Extraction. In this stage, EVS collect sensor readings when each event occurs. R-EVS extract rules from the collected traces in the form of statistical correlations between an event (`E`) and its influence on sensor readings (`S`). For example, R-EVS learn that when a light turns on, the readings of two illuminance sensors change, and extract the following physical fingerprint: $\texttt{Rule}:\texttt{light-on} \leftrightarrow \{S_1.\texttt{Illum} = \texttt{High}, S_2.\texttt{Illum} = \texttt{High}\}$. R-EVS often map numerical sensor measurements into binary (high/low or increase/decrease) or categorical (high/medium/low) values since environmental factors may introduce noise in the numerical sensor readings.

ML-EVS learn a physical fingerprint model for each event using features extracted from sensor readings. For instance, a `light-on` model is learned based on the features (e.g., min,

max, mean, sum, stdev) extracted from numerical illuminance sensor readings. To account for environmental noise, ML-EVS pre-process the sensor data by applying smoothing filters and signal processing (e.g. moving average) for noise reduction.

Physical Fingerprint Checking. EVS use physical fingerprints to detect event spoofing and masking attacks at run-time. R-EVS check if the event's rule is satisfied when an event notification is received. For instance, when a `light-on` notification is received, R-EVS check if the illuminance sensors measure the expected high readings. If the sensor readings deviate from what rules dictate, they deem `light-on` event has not physically occurred and flag a spoofing attack. To detect event masking, R-EVS continuously monitor the sensor readings and check if they map to an event's rule. If the sensor readings match an event's rule, but an event notification is not received, they flag a masking attack. For instance, if the sound sensors measure high ambient sound but an `alarm-on` event notification is not received, they detect an event masking attack.

To detect event spoofing, ML-EVS predict if a physical event occurred or not from runtime sensor readings based on the learned event model. If a notification is received but the model predicts the event has not occurred, ML-EVS flag a spoofing attack. For event masking, they continuously evaluate each event model with the features extracted from sensor readings. If the event model predicts a physical event has occurred, yet, its notification is not received, ML-EVS flag a masking attack.

III. Threat Model

EVS, in their threat model, assume that an adversary can conduct *event spoofing* and *event masking* attacks [11], [12], [34], [67], [68]. The adversary can conduct such attacks through (1) phantom devices or (2) malicious apps. First, a phantom device is a computer program that mimics a real physical device [87]. It enables the adversary to remotely communicate with the IoT hub using credentials (device ID and legitimacy information) stolen from a real device. An adversary can steal this information through (1) public data from company websites (e.g., Github repositories) [83] and (2) vulnerable IoT apps [84]. Second, the adversary can use malicious IoT apps that exploit the design vulnerabilities in IoT programming frameworks [31], [75]. These apps subvert the intended use of programming APIs to create legitimate event objects and communicate with the IoT hub. In these attacks, the adversary does not inject commands to physically control devices, but stealthily perform remote event spoofing or masking.

Prior works on EVS also assume adversaries have no physical access to the devices, and the sensors used in EVS for attack detection are trusted. The sensors are trusted due to two main reasons. First, many sensors send periodic readings to the hub, which requires the adversary to *continuously* mask the real sensor's readings and spoof the injected values. Second, there have been various systems proposed to detect sensor attacks [12], [25], [30], [81]. Yet, we note that if an adversary is able to spoof *all* events and sensor readings, all attacks become trivial as the adversary can evade both event verification and sensor spoofing detection systems.

The adversary's goal is to evade EVS by spoofing and masking events without getting detected. For this, we assume the adversary passively sniffs the smart home communications without intercepting or injecting any messages. The adversary can recognize events in real-time over unencrypted or encrypted device communication through existing IoT network analysis tools [2], [8], [40], [71], [85]. This allows the adversary to determine the time to conduct the attacks, maximizing their likelihood to evade EVS. To determine which event to spoof or mask, an adversary can predict events' physical influences on sensor measurements and spoof or mask events accordingly, or a knowledgeable adversary who knows the physical event fingerprints of EVS can smartly select the event (Detailed in Section IV). Our knowledgeable adversary is similar to the opportunistic attacker in [11], [12] that knows EVS parameters and identifies opportunities to evade EVS. Here, the opportunistic attacker aims to minimize the time they have to wait before spoofing or masking, while we do not bound the adversary's wait time. This is because the adversary can conduct remote attacks with minimal effort.

IV. Motivation

Event spoofing and masking attacks create inconsistencies between the physical and cyber device states. By leveraging these inconsistencies, an adversary can indirectly gain control over safety-critical devices and prevent invoking the user's intended device actuations, putting the user and environment at risk. Therefore, there is a growing interest in EVS to detect event spoofing and masking attacks through physical fingerprints [4], [11], [12], [25], [34], [58], [64], [67], [68].

EVS infer the physical event fingerprints that describe the relations between events and sensor readings to detect anomalies in physical and cyber device states. However, complex physical relations exist between events and sensor readings [19], [27], [56]: (1) An event may influence single or multiple physical channels (e.g., the `TV-on` event introduces sound and illuminance), (2) a sensor measures the influence of single or multiple events (e.g., an illuminance sensor measures the aggregated illuminance from `TV-on` and `light-on`), and (3) when the distance between an actuator and sensor increases, the event's influence on sensor readings monotonically decreases.

We argue that to properly detect the event spoofing and masking attacks, such physical properties must be learned, incorporated, and assessed in EVS. Unfortunately, state-of-the-art EVS [11], [12], [25], [34], [67], [68] broadly ignore this requirement, enabling an adversary to evade EVS, launching successful spoofing and masking attacks. Below, we detail three evasion attacks that exploit the physical properties between events and sensor readings to bypass EVS given the same threat model. We introduce a set of techniques in Section V that require the interplay of physical fingerprint analysis, software patching, and sensor placement to discover evadable fingerprints and mitigate them via robust fingerprints.

Evasion$_1$: Fingerprint Encapsulation (Figure 2(a)). If an event's influence on sensor measurements satisfies the fingerprint of another event, the adversary can conduct a spoofing attack when the physical event occurs at run-time and evade EVS. For instance, in Figure 2(a), EVS learn that the `TV-on` event influences the readings of the sound (`S1`) and illuminance (`S2`) sensors, and the `light-on` event similarly influences `S2`'s readings. At run-time, when `TV-on` physically occurs, the sensors measure its influence. At this time, if the adversary spoofs `light-on` when its physical state is `off`, EVS cannot detect

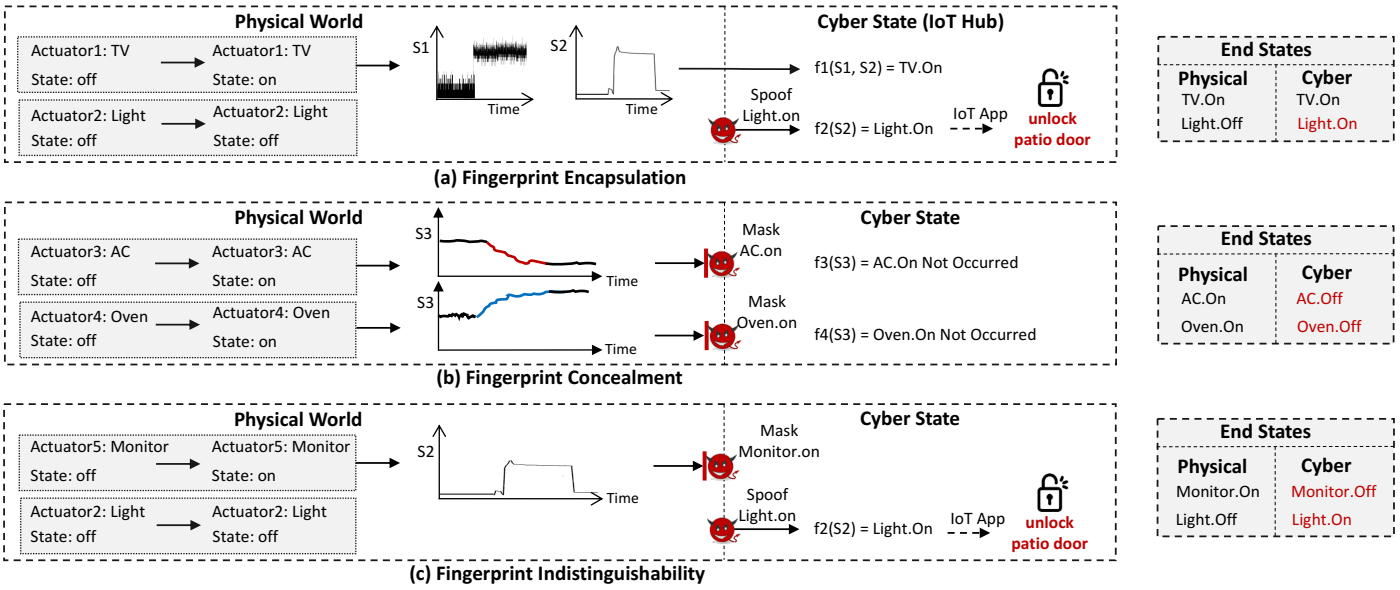


Fig. 2: Illustration of the evasion attacks against Event Verification Systems

this attack as S2 measures the influence from TV-on. Thus, the IoT hub thinks that the light's cyber state is on; yet, its physical state is off. Through this, the adversary can trigger apps conditioned on light-on, e.g., an app that unlocks the patio door, assuming the user arrived home.

This attack's root cause is that an event's (E_i) influence on sensors encapsulates another event's (E_j) influence. Formally, R-EVS are vulnerable to this attack if there are two rules:

$$\texttt{Rule}_1 : E_i \leftrightarrow \{S_{i,1} = \texttt{High}, S_{i,2} = \texttt{High}, ..., S_{i,n} = \texttt{High}\}$$
$$\texttt{Rule}_2 : E_j \leftrightarrow \{S_{j,1} = \texttt{High}, S_{j,2} = \texttt{High}, ..., S_{j,m} = \texttt{High}\}$$
$$\{S_{j,1}, S_{j,2}, ..., S_{j,m}\} \subseteq \{S_{i,1}, S_{i,2}, ..., S_{i,n}\}$$

Here, $S_{a,b}$ indicates the b^{th} sensor's measurement when event a occurs. In ML-EVS, this attack happens if the fingerprint model for E_j predicts E_j has occurred when E_i physically occurs.

Evasion$_2$: Fingerprint Concealment (Figure 2(b)). Two events may have *opposing* influences on one or more sensor's readings; one event increases and another event decreases them. For instance, in Figure 2(b), AC-on decreases the temperature sensor's readings whereas oven-on increases them. When these events occur, they conceal each other's impact on sensor readings and prevent EVS from detecting masking attacks launched on one or both of these events. Through this, an IoT hub thinks physically occurred events have not occurred. This prevents triggering the IoT apps conditioned on these events. For instance, it may prevent an app that turns off the oven as the oven's cyber state is already off, and create a fire hazard [3].

This attack occurs in R-EVS if there are two rules that include the following expected sensor readings.

$$E_i \leftrightarrow S = \texttt{Inc} \text{ and } E_j \leftrightarrow S = \texttt{Dec}$$

When E_i and E_j occur, the sensor S may not measure any increase (Inc) or decrease (Dec). When the adversary masks any of these events' notifications, R-EVS cannot detect it as an expected sensor value in the rule is not satisfied. Although there could be multiple sensors that measure the events, R-EVS often do not raise alarms when their rules are partially satisfied to minimize false alarms. In ML-EVS, this attack occurs if one or both event's model outputs event-not-occurred when the events physically occur due to their opposing influence.

Evasion$_3$: Fingerprint Indistinguishability (Figure 2(c)). When two events have similar physical fingerprints, an adversary can mask a physically occurred event's notification and spoof the other event which has not occurred. We call this attack *mask-and-spoof*. For instance, in Figure 2(c), both monitor-on and light-on events only influence the illuminance sensor's readings. When monitor-on occurs, the adversary masks its notification and spoofs light-on. EVS cannot detect this as the illuminance sensor measures the monitor's influence. Thus, the IoT hub thinks the light is on and the monitor is off, but in the physical world, the light is off and the monitor is on. Through this, the adversary can prevent triggering the apps conditioned on monitor-on, and trigger the ones conditioned on light-on instead, such as an app that unlocks the patio door.

This attack occurs when two events have the same rules, preventing the R-EVS from distinguishing between those events.

$$\texttt{Rule}_1 : E_i \leftrightarrow \{S_{i,1} = \texttt{High}, S_{i,2} = \texttt{High}, ..., S_{i,n} = \texttt{High}\}$$
$$\texttt{Rule}_2 : E_j \leftrightarrow \{S_{j,1} = \texttt{High}, S_{j,2} = \texttt{High}, ..., S_{j,m} = \texttt{High}\}$$
$$\{S_{j,1}, S_{j,2}, ..., S_{j,m}\} = \{S_{i,1}, S_{i,2}, ..., S_{i,n}\}$$

ML-EVS are vulnerable to this attack if an event's model (E_j) outputs E_j-occurred when another event (E_i) physically occurs, and E_i's model outputs E_i-occurred when E_j occurs, making the events indistinguishable to ML-EVS.

Aggregated Evasion Attacks. We have presented three evasion attacks for EVS that stem from an event's physical fingerprint being similar or opposing to a single event. Yet, in complex smart homes with diverse sensors and actuators, the aggregated (joint) influence of multiple events on sensor readings may also satisfy or conceal an event's fingerprint and enable an adversary to conduct evasion attacks. For instance, an event (E_k) is vulnerable to aggregated $\texttt{Evasion}_1$ (spoofing attack) if its rule is encapsulated by two or more events (E_i, E_j).

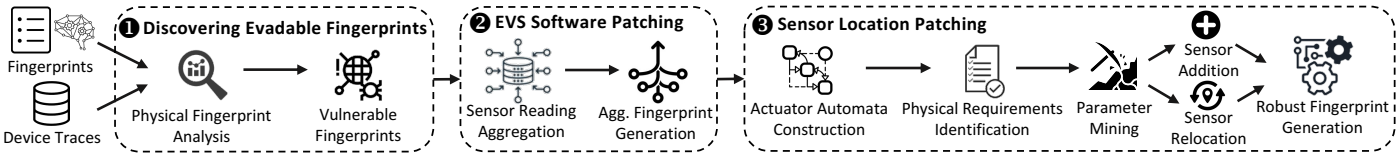


Fig. 3: Overview of physical event fingerprint patching

$$\texttt{Rule}_1 : \texttt{E}_\texttt{i} \leftrightarrow \{\texttt{S}_{\texttt{i},1} = \texttt{High}, \texttt{S}_{\texttt{i},2} = \texttt{High}, ..., \texttt{S}_{\texttt{i},\texttt{n}} = \texttt{High}\}$$
$$\texttt{Rule}_2 : \texttt{E}_\texttt{j} \leftrightarrow \{\texttt{S}_{\texttt{j},1} = \texttt{High}, \texttt{S}_{\texttt{j},2} = \texttt{High}, ..., \texttt{S}_{\texttt{j},\texttt{m}} = \texttt{High}\}$$
$$\texttt{Rule}_3 : \texttt{E}_\texttt{k} \leftrightarrow \{\texttt{S}_{\texttt{k},1} = \texttt{High}, \texttt{S}_{\texttt{k},2} = \texttt{High}, ..., \texttt{S}_{\texttt{k},\texttt{l}} = \texttt{High}\}$$
$$\{\texttt{S}_{\texttt{k},1}, \texttt{S}_{\texttt{k},2}, ..., \texttt{S}_{\texttt{k},\texttt{l}}\} \subseteq \{\texttt{S}_{\texttt{i},1}, \texttt{S}_{\texttt{i},2}, ..., \texttt{S}_{\texttt{i},\texttt{n}}\} \cup \{\texttt{S}_{\texttt{j},1}, \texttt{S}_{\texttt{j},2}, ..., \texttt{S}_{\texttt{j},\texttt{m}}\}$$

In ML-EVS, an event model ($\texttt{E}_\texttt{k}$) is vulnerable to aggregated $\texttt{Evasion}_1$ attack if its model predicts `event-occurred` when other events ($\texttt{E}_\texttt{i}$, $\texttt{E}_\texttt{j}$) physically occur. Similarly, the aggregated influences from multiple events may enable the adversary to conduct $\texttt{Evasion}_2$ and $\texttt{Evasion}_3$ attacks.

V. Robust Physical Fingerprint Generation

We identify the evadable physical fingerprints in EVS and fix them by patching the EVS software. When the software patching fails, we identify the sensor locations for robust fingerprint generation, preventing attackers from evading EVS.

Approach Overview. Our approach to robust physical fingerprint generation is illustrated in Figure 3.

Given the event fingerprints generated by an EVS and collected event and sensor traces, we first discover the fingerprints that are vulnerable to evasion by checking if a fingerprint can be satisfied or concealed due to other events' influences (❶).

We fix evadable event fingerprints with two complementary methods: EVS software patching and sensor location patching. Software patching is the process of creating additional fingerprints that define the relation between multiple events and their joint influence on sensors (❷). We define these new fingerprints in the EVS software to prevent $\texttt{Evasion}_1$ and $\texttt{Evasion}_2$ attacks. Yet, software patching cannot prevent $\texttt{Evasion}_3$ attacks since they occur when the events' fingerprints are indistinguishable, and thus, even the new fingerprints fail to detect this attack.

To address the limitations of software patching, we introduce sensor location patching, which prevents evasion attacks through an appropriate sensor placement, ensuring events have a unique physical fingerprint (❸). For this, we develop a technique that models actuators with hybrid automata, and extends parameter mining to derive the angles and distances of sensors from the actuators to generate robust physical fingerprints.

System Deployment. Software patching is an automated process that creates new physical fingerprints by analyzing the existing traces of EVS, and integrates them into EVS software.

Sensor location patching is a security-by-design approach that complements software patching when unique physical fingerprints are not found. It requires the actuator locations in the smart home as an input. We determine the locations using Lumos [65], an IoT device localization tool, which only requires the user to walk within the home with a mobile phone or tablet. Sensor location patching outputs sensor placement regions that guarantee events have unique physical fingerprints.

Algorithm 1 Discovering Evadable Physical Fingerprints

Input: Event List ($\texttt{L}_\texttt{e}$), Sensor List ($\texttt{L}_\texttt{s}$), Event Rules ($\texttt{Rule}_\texttt{e}$), Event Models ($\texttt{f}_\texttt{e}(\texttt{X})$ where $\texttt{X}$ is an event's feature matrix)
Output: Evadable Fingerprints ($\texttt{L}_{evade}$)

1: **function** VulnerabilityAnalysis($\texttt{L}_\texttt{e}, \texttt{L}_\texttt{s}, \texttt{Rule}_\texttt{e}, \texttt{f}(\texttt{e})$)
2: **for** $\texttt{E}_\texttt{i} \in \texttt{L}_\texttt{e}$, $\texttt{E}_\texttt{in} \subset \texttt{L}_\texttt{e}$ **do**
3: **if** $\texttt{Rule}_{\texttt{E}_\texttt{i}} \subseteq \texttt{Agg}_\texttt{r}(\texttt{Rule}_{\texttt{E}_\texttt{in}})$ **then** $\texttt{L}_{evade} \leftarrow \texttt{L}_{evade} \cup \texttt{Rule}_{\texttt{E}_\texttt{i}}$
4: **end if**
5: **if** $\texttt{f}_{\texttt{E}_\texttt{i}}(\texttt{Agg}_\texttt{f}(\texttt{X}_{\texttt{E}_\texttt{in}})) = 1$ **then** $\texttt{L}_{evade} \leftarrow \texttt{L}_{evade} \cup \texttt{f}_{\texttt{E}_\texttt{i}}$
6: **end if**
7: **if** $\texttt{Opp}(\texttt{Rule}_{\texttt{E}_\texttt{i}}, \texttt{Rule}_{\texttt{E}_\texttt{in}})$ **then** $\texttt{L}_{evade} \leftarrow \texttt{L}_{evade} \cup \texttt{Rule}_{\texttt{E}_\texttt{i}}$
8: **end if**
9: **if** $\texttt{f}_{\texttt{E}_\texttt{i}}(\texttt{Agg}_\texttt{f}(\texttt{X}_{\texttt{E}_\texttt{i}}, \texttt{X}_{\texttt{E}_\texttt{in}})) = 0$ **then** $\texttt{L}_{evade} \leftarrow \texttt{L}_{evade} \cup \texttt{f}_{\texttt{E}_\texttt{i}}$
10: **end if**
11: **end for**
12: **return** $\texttt{L}_{evade}$
13: **end function**

IoT service providers either relocate or add a minimal number of sensors to the identified regions. To reflect the changes from these sensors, we revise the EVS fingerprints to prevent the evasion attacks that software patching could not fix.

Our defenses support added, updated, or removed devices. If a new device is introduced or an existing device is updated (e.g., a sensor's threshold is changed), EVS conduct data collection with these devices to update the existing fingerprints and generate new ones. In these cases, our defenses must be re-used to protect them against evasion. If an actuator is removed, EVS remove its fingerprint, and if a sensor is removed, EVS update the fingerprints extracted from that sensor. Here, sensor removals may create a vulnerability to evasion if that sensor was distinguishing an event, and thus, our defenses must be re-used after sensor removals. In practice, our defenses do not introduce a significant additional effort compared to EVS while making them secure against evasion.

A. Discovering Evadable Fingerprints

We first discover the fingerprints of EVS that are vulnerable to evasion attacks, as shown in Algorithm 1.

1) Fingerprint Satisfaction Analysis: We analyze if an event's fingerprint is satisfied when other events occur to detect $\texttt{Evasion}_1$ and $\texttt{Evasion}_3$ vulnerabilities. In R-EVS, we identify vulnerable rules by checking if other individual or aggregated event rules are the same or encapsulate an event's rule (Line 3-4). For this, we define a rule aggregation function ($\texttt{Agg}_\texttt{r}$) that takes, as input, individual event rules and outputs their aggregation. If two events influence separate sensors, $\texttt{Agg}_\texttt{r}$ appends their rules. If two events influence the same sensor's readings, $\texttt{Agg}_\texttt{r}$ sums the numerical ranges of their categories.

In ML-EVS, we identify the vulnerable event models by checking if the features extracted from other individual events or their aggregations make the model evaluate to `event-occurred`

(Line 5-6). To find the aggregations of the features, we define an $\texttt{Agg}_f$ function that takes as input the features extracted from each individual event and outputs their aggregation.

2) Fingerprint Concealment Analysis: We analyze whether an event's fingerprint is concealed when other events occur to detect $\texttt{Evasion}_2$ vulnerabilities. In R-EVS, we identify an event's rule as vulnerable if another event or multiple events' aggregation influences a sensor's readings in opposing ways (Line 7-8). We define the `Opp` function, which takes multiple rules as input and outputs if the first rule is opposing to the others. In ML-EVS, we identify the vulnerable event models by checking if the aggregated features of the analyzed event with other events evaluate to `event-not-occurred` (Line 9-10). If it does, this indicates when these events occur and the adversary masks the analyzed event's notification, ML-EVS cannot detect this as its model does not indicate the event has occurred.

B. EVS Software Patching

We craft new fingerprints that make EVS dynamically adapt the expected sensor measurements based on the other events that are physically occurring in the smart home.

1) Sensor Data Aggregation: We aggregate the sensor readings collected at EVS data collection to determine the expected sensor readings when multiple events occur. We leverage these readings to learn aggregated fingerprints robust against evasion. One may consider using the computed aggregated rules and features to learn them. Yet, in R-EVS, we need to adjust the categories based on events' aggregated influences. In ML-EVS, we require training data to learn aggregated event models. To address this, we define a sensor reading aggregation operator ($\texttt{Agg}_s$) that computes the joint influence of events on a sensor.

The sensor reading aggregations are often computed with linear addition [86]. Yet, sound pressure is measured in decibels that cannot be aggregated with addition. We define $\texttt{Agg}_s$ for sound as $\texttt{Agg} = 10 \times \log_{10}(\sum_{i=1}^{n} 10^{L_i/10})$ where $L_i, i \in n$ represents the sound pressures [53]. Besides, if a sensor outputs boolean-typed readings (e.g., `sound-detected`), we define $\texttt{Agg}_s$ as the "or" operator, which outputs 1 if an event's influence on the sensor is 1, and 0 otherwise.

2) Aggregated Fingerprint Generation: We generate new fingerprints from the aggregated sensor measurements that define the expected sensor measurements when multiple events occur. From these, EVS dynamically adjust their fingerprints based on physically occurring events and avoid evasion.

Generating Aggregated Rules. Run-time rule checks for R-EVS can be implemented with a set of if statements. A generic rule ($\texttt{Rule}_1 : E_1 \leftrightarrow \{S_1 = \texttt{High}, S_2 = \texttt{Med}\}$) can be implemented with the following code block, where the `High` category for S_1 is > 100 and `Med` category for S_2 is $[50, 100]$.

```
1 m1 = sensor1.read()
2 m2 = sensor2.read()
3 if m1 < 100 or (m2 < 50 or m2 > 100):
4     print("Spoofing Attack Detected")
```

This check is performed within a time window (w) after E_1 notification is received to detect spoofing. If run-time sensor readings deviate from the expected values, an attack is detected.

If this rule is vulnerable to evasion as it is encapsulated by another event's rule (e.g., $\texttt{Rule}_2 : E_2 \leftrightarrow \{S_1 = \texttt{High}, S_2 = \texttt{Med}, S_3 = \texttt{High}\}$), we generate a new rule that defines their aggregation. The aggregated rule's S_2 measurement can be determined as `High`. For S_1, both events' rules indicate `High`. Yet, if we define the aggregated rule's expected reading as `High`, this cannot distinguish between aggregated and individual events. To address this, we define another category if the aggregated rule's category is identical to individual rules but numerically different. For instance, if E_1's influence on S_1 is within $[100, 150]$, E_2's influence is within $[100, 200]$, and their aggregated influence is within $[200, 350]$, we define `High` category as $[100, 200]$ and `Agg_High` as > 200. The patched rule for E_1 is presented below.

```
1 m1 = sensor1.read()
2 m2 = sensor2.read()
3 m3 = sensor3.read()
4 if E2 == 0 and (m1 < 100 or (m2 < 50 or m2 > 100)):
5     print("Spoofing Attack Detected")
6 elif E2 == 1 and (m1 < 200 or m2 < 100 or m3 < 100)
7     print("Spoofing Attack Detected")
```

If E_2 is not occurring, the rule check does not change (Line 4). If E_2 occurs, R-EVS check the aggregated rule (Line 6). If the adversary spoofs the E_1 event when E_2 occurs, the patched rule gets violated at Line 6 as the sensors are only measuring E_2's influence. Similarly, we leverage the aggregated rules to prevent $\texttt{Evasion}_2$ attacks. If two events have an opposing influence on a sensor's measurements, that sensor is removed from the aggregated rule to prevent evasion.

Generating Aggregated Models. For vulnerable events, we derive aggregated models ($\texttt{f}_{E_1,\ldots E_n}(x)$) using the features from aggregated sensor readings. Similar to individual models, the aggregated models have two classes: `events-occurred` and `events-not-occurred`. To detect spoofing, ML-EVS dynamically decide if they should check the individual or aggregated model based on received notifications. For instance, if E_1's notification arrives, ML-EVS check $\texttt{f}_{E_1}(x) = 1$. If both E_1's and E_2's notifications arrive, they check $\texttt{f}_{E_1,E_2}(x) = 1$. To detect masking, ML-EVS periodically check if the aggregated models output `events-occurred` but no event notification is received.

3) Limitations of EVS *Software Patching:* Fingerprint indistinguishability ($\texttt{Evasion}_3$) and certain concealment ($\texttt{Evasion}_2$) attacks where two or more events completely conceal each other's influence cannot be prevented with software patching. First, if events have indistinguishable fingerprints and the adversary conducts a mask-and-spoof attack, the software-patched EVS cannot differentiate which event has occurred, and cannot detect this attack. Second, if events completely conceal their influence, the sensor readings change negligibly. Thus, when the adversary masks their notifications, EVS cannot detect this. Additionally, software patching cannot be used in sensors with boolean-typed attributes (e.g., `sound-detected`) or if sensor saturation occurs (e.g., humidity sensor reaches 100%). In such cases, the aggregated influence from multiple events becomes indistinguishable from the event's individual influence, where the sensor outputs a boolean or min/max value.

C. Sensor Location Patching

We introduce sensor location patching to prevent the evasion attacks that software patching cannot address. We determine the sensor placement regions that ensure physical fingerprints are unique for each event. After a minimal number of sensors are added or relocated to the identified regions, we extract robust physical fingerprints, which can detect all evasion attacks.

1) Actuator Automata Construction: We construct hybrid automata to model an event's influence on each physical channel. The automata enable exhaustively and efficiently testing the events' influences at different locations to determine the sensor placements. Although one may consider conducting physical experiments to determine the sensor placement, multiple tests with different placements are impractical due to the substantial effort required. Below, we map each event to the physical channels they influence and construct an automaton for each.

Mapping Events to Physical Channels. We leverage the EVS fingerprints to obtain the physical channels that an event influences, and build an *event-physical channel mapping*. This mapping is a binary matrix, $\texttt{Map}[\texttt{p}, \texttt{e}]$, where $\texttt{p}$ is a physical channel and $\texttt{e}$ is an event. $\texttt{Map}[\texttt{p}, \texttt{e}] = 1$ means $\texttt{e}$ influences $\texttt{p}$, and 0 means it does not. The mapping enables us to construct an actuator automaton for each physical channel it influences.

Constructing the Automata. Actuator automata model an event's influence on a physical channel over time at different locations. An actuator's event may influence multiple physical channels; therefore, we construct an automaton for each event and channel identified through their mapping. Formally, we model an actuator as a hybrid automaton [44], $\texttt{H} = (\texttt{Q}, \texttt{X}, \texttt{f}, \rightarrow)$. Here, $\texttt{Q}$ is the actuator's discrete states (e.g., on/off) and $\texttt{X}$ is a continuous variable that defines the event's influence on a channel (e.g., change in temperature). The flow function ($\texttt{f}$) takes three parameters, the distance from the actuator, min/max output values and device property parameter that describes the actuator characteristics (e.g., its light intensity), and computes $\texttt{X}$ over time. The flow functions are unique to each physical channel. For continuous influences (e.g., temperature), they are defined with differential equations and for instant influences (e.g., sound), they are defined with algebraic equations.

Overall, we study 12 actuators and three physical channels commonly used in EVS to construct 18 automata (an event may impact multiple physical channels) considered in our evaluation. We detail their implementations in Appendix C. We define generic flow functions for each physical channel by leveraging well-studied equations from control theory [42], [56], [74], [86]. We next set the parameters in the flow functions. First, we set different values to the distance parameter to test different possible sensor locations (Detailed in Section V-C3). Second, we set the min/max output parameters based on the highest and lowest measurements a sensor can output from sensor datasheets to handle sensor saturation and ensure the automata do not output values outside these ranges.

To set the device property parameter, we leverage the (τ, ϵ)-closeness [1] metric that measures the fidelity of the automata with actual device traces in their timings (τ) and values (ϵ). We use the data collected for EVS fingerprint generation to determine the device property parameters that maximize the (τ, ϵ)-closeness of our automata with real device traces. Particularly, we execute the automata with a binary search on the device property parameter and collect automata traces that represent an event's influence on a physical channel. We then obtain the optimal parameter value that minimizes the deviation (ϵ) between automata and real device traces. This process ensures the constructed automata have high fidelity with the real devices used in smart homes.

2) Physical Requirements Identification: We observe that to prevent the evasion attacks, a sensor must be placed in a way that it measures a single event's influence. This ensures the fingerprint extracted from that sensor is unique. Following this, we formalize the requirements that minimize the number of events impacting the same sensor's readings while maximizing the number of sensors detecting an event's influence. We represent the requirements with linear temporal logic (LTL) to formally reason about them on the automata and derive a sensor placement that prevents the evasion attacks.

Algorithm 2 Physical Requirements Formalization

Input: Event List ($\texttt{L}_\texttt{e}$), Physical Channel List ($\texttt{L}_\texttt{p}$), Detection Thresholds ($\tau_\texttt{p}$), Event - Physical Channel Mapping ($\texttt{Map}[\texttt{p}, \texttt{e}]$)
Output: LTL Constraint List for each model ($\texttt{L}_{\phi_{\texttt{p},\texttt{e}}}$)

1: **function** REQ_MODELING($\texttt{L}_\texttt{e}, \texttt{L}_\texttt{p}, \tau_\texttt{p}, \texttt{Map}[\texttt{p}, \texttt{e}]$)
2: $\quad\texttt{L}_{\phi_{\texttt{p},\texttt{e}}} = \emptyset$
3: $\quad$**for** $\texttt{j} \in \texttt{L}_\texttt{p}$ **do**
4: $\quad\quad\texttt{N} = \texttt{Sum}(\texttt{Map}[\texttt{i}, :])$
5: $\quad\quad$**for** $\texttt{i} \in \texttt{L}_\texttt{e}$ **do**
6: $\quad\quad\quad$**if** $\texttt{Map}[\texttt{i}, \texttt{j}] == 0$ **then** Break
7: $\quad\quad\quad$**else**
8: $\quad\quad\quad\quad\phi_{\texttt{i},\texttt{j}} = \square(\texttt{imp}(\langle \texttt{i}, \texttt{j} \rangle) > \tau_\texttt{i})$
9: $\quad\quad\quad\quad\bar{\phi}_{\texttt{i},\texttt{j}} = \square(\texttt{imp}(\langle \texttt{i}, \texttt{j} \rangle) < \tau_\texttt{i}/(\texttt{N}-1))$
10: $\quad\quad\quad$**end if**
11: $\quad\quad\quad\texttt{L}_{\phi_{\texttt{p},\texttt{e}}} = \texttt{L}_{\phi_{\texttt{p},\texttt{e}}} \cup \phi_{\texttt{i},\texttt{j}} \cup \bar{\phi}_{\texttt{i},\texttt{j}}$
12: $\quad\quad$**end for**
13: $\quad$**end for**
14: $\quad$**return** $\texttt{L}_{\phi_{\texttt{p},\texttt{e}}}$
15: **end function**

Algorithm 2 details the steps for deriving a list of LTL formulas. We use these formulas to derive each event's *maximal* and *minimal influence regions*, which define the locations where an event's influence is always detected by a sensor, and never detected by a sensor, respectively. For each physical channel, constraint formalization first finds the number of events that influence the same physical channel (Line 4). It then defines two LTL formulas for each event (Lines 8-9).

Maximal Influence Formula. The first LTL formula (Line 8) ensures that an event's ($\texttt{e}$) influence on a physical channel ($\texttt{p}$) always exceeds a threshold (τ):

$$\square(\texttt{imp}(\langle \texttt{e}, \texttt{p} \rangle) > \tau)$$

Here, $\square$ means always. Since an event's influence on sensor readings depends on distance, this formula's satisfaction depends on the distance between the sensor and event. The threshold defines the minimum influence required for an event to make a change in sensor readings. We determine the thresholds based on the sensor's sensitivity level defined in its datasheet.

Minimal Influence Formula. The second formula (Line 9) allows us to derive the minimal influence regions. For this, we also consider the aggregated influences from multiple events. We tailor the formula so that the aggregation of multiple events' influence does not exceed the sensor threshold.

$$\square(\texttt{imp}(\langle \texttt{e}_2, \texttt{p} \rangle) + \ldots + \texttt{imp}(\langle \texttt{e}_\texttt{N}, \texttt{p} \rangle) < \tau)$$

Yet, this formula has high complexity since it includes physical impacts from multiple events. To address this, we separate the minimal influence formula to individual events by adjusting the threshold. We leverage the fact that if individual influences do not exceed $\tau/(\texttt{N}-1)$, where $\texttt{N}$ is the number of events impacting the same channel (derived in Line 4), their aggregation cannot exceed τ. Thus, we define the formula as

$$\square(\texttt{imp}(\langle \texttt{e}_\texttt{i}, \texttt{p} \rangle) < \tau/(\texttt{N}-1))$$

Algorithm 3 Sensor Location Identificatio

Input: Event List ($\mathtt{L_e}$), Physical Channels ($\mathtt{L_p}$), Actu
LTL Formulas ($\mathtt{L}_{\phi_{\mathtt{p,e}}}$), Automata ($\mathtt{H_{p,e}}$)
Output: Sensor Location Regions ($\mathtt{L_0}$)

1: **function** LOCATIONGENERATION($\mathtt{L_e}, \mathtt{L_p}, \mathtt{E_{x,y,z}},$
2: $\mathtt{L_0} = \emptyset$
3: **for** $\mathtt{i} \in \mathtt{L_p}$ **do**
4: $\mathtt{ct} = 0$
5: **for** $\mathtt{j} \in \mathtt{L_e}$ **do**
6: **if** $\phi_{i,j} \neq \emptyset$ **then**
7: $\mathtt{r_{i,ct}} = \mathtt{Parameter\_Mining}(\mathtt{H_{i,j}},$
8: $\mathtt{C_{i,ct}}: (\mathtt{x} - \mathtt{i_x})^2 + (\mathtt{y} - \mathtt{i_y})^2 +$
9: $\bar{\mathtt{r}}_{\mathtt{i,ct}} = \mathtt{Parameter\_Mining}(\mathtt{H_{i,j}},$
10: $\bar{\mathtt{C}}_{\mathtt{i,ct}}: (\mathtt{x} - \mathtt{i_x})^2 + (\mathtt{y} - \mathtt{i_y})^2 +$
11: $\mathtt{ct} = \mathtt{ct} + 1$
12: **end if**
13: **end for**
14: **for** $\mathtt{j} = 0 : \mathtt{ct} - 1$ **do**
15: $\mathtt{Reg} = \mathtt{C_{i,j}} \setminus \{\bar{\mathtt{C}}_{\mathtt{i,j}}\}, \mathtt{k} = 0 : \mathtt{ct} - 1, \mathtt{k} \neq \mathtt{j}$
16: **if** $\mathtt{Reg} = \emptyset$ **then**
17: **for** $\mathtt{Loc} \in \mathtt{C_{i,j}}$ **do**
18: $\mathtt{G} = \mathtt{imp(i,j)}/(\sum_{\mathtt{k}=0}^{\mathtt{k}=\mathtt{ct}-1,\mathtt{k}\neq\mathtt{j}}(\mathtt{imp(i,k)})$
19: **end for**
20: $\mathtt{L_0} = \mathtt{L_0} \cup \langle \mathtt{p}, \mathtt{G} \rangle$
21: **else** $\mathtt{L_0} = \mathtt{L_0} \cup \langle \mathtt{p}, \mathtt{Reg} \rangle$
22: **end if**
23: **end for**
24: **end for**
25: **return** $\mathtt{L_0}$
26: **end function**

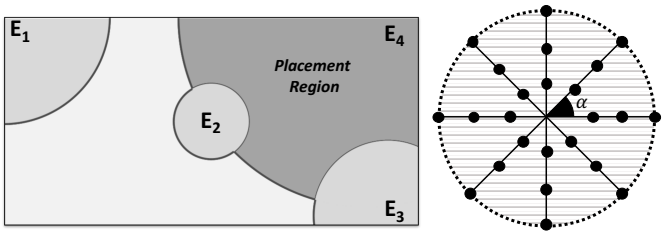
This guarantees even if $\mathtt{N} - 1$ events' influences aggregate, they cannot have a detectable influence on sensors.

3) Sensor Location Identification: We introduce Algorithm 3, which outputs sensor placement regions, given the actuators' locations, automata constructed in Section V-C1, and LTL formulas derived with Algorithm 2. To learn the actuators' locations, we use Lumos [65], an IoT device localization tool. Lumos locates devices with high precision using mobile device sensors and wireless signal strength measurements.

When a sensor is placed in an identified region, the fingerprint extracted from it ensures that the event has a unique physical signature. Since we formalize the physical requirements that ensure each event's fingerprint is unique with LTL and conduct formal analysis (parameter mining and circular grid search) to identify sensor placement regions that satisfy the LTL formulas, sensor location patching formally guarantees evasion attacks are prevented under the assumption of actuator automata correctness.

Robustness-guided Parameter Mining. For each physical channel and event, we compute the boundary distance from the event that satisfies the LTL formulas through parameter mining [45]. Parameter mining conducts a deterministic or stochastic search to derive a parameter value that incurs a robustness of ≈ 0 (i.e., "barely" satisfies the formula). The robustness is the LTL formula's satisfaction degree on the automata, where positive robustness values indicate LTL formula is satisfied, and negative values indicate it is not.

We conduct parameter mining on the distance parameter in the actuator automata to compute the maximal and minimal influence regions of each event (Line 7-10). *Maximal influence region* is the area where the event's physical influence is always detectable by the sensor, and *minimal influence region* is the area it is never detectable. We then consider each event as the "target event", which is the event to place a sensor that detects its physical influence. We take the difference of the target event's maximal influence region from the minimal influence regions of others to find the placement region where the sensor only measures the target event's influence (Line 15).



(a) Parameter mining output (b) Grid search

Fig. 4: Algorithm 3's sample output. (a) Four actuators' events influence the same physical channel, and a sensor can be placed in the dark area with $\mathtt{E_4}$ as the target event, (b) 2D representation of the 3D circular grid search.

An example output of Algorithm 3 is presented in Figure 4a, where four actuators' events $\mathtt{E_1}$-$\mathtt{E_4}$ impact the same physical channel and the target event to place the sensor for is $\mathtt{E_4}$. We repeat this for all event-physical channel pairs to maximize the number of sensors that measure an event's influences.

Circular Grid Search. Parameter mining finds the sensor placement regions where only the target event influences the sensor readings. Yet, there may be cases that a clear region with no influence from other events do not exist (Line 16) due to actuator placements and high influences from some events relative to others. To address this, we conduct a *3D circular grid search* within the target event's maximal influence region to find the locations where the target event has the highest influence compared to others (Lines 17-19). This ensures the fingerprint extracted from the sensor can distinguish the target event, preventing the evasion attacks. We conduct the search as $0° : \alpha° : (360 - \alpha)°$, $0° : \beta° : (360 - \beta)°$ and $\mathtt{s} \times \mathtt{r} : \mathtt{s} \times \mathtt{r} : \mathtt{r}$, where α and β are angle steps, $\mathtt{s}$ is the distance step, and $\mathtt{r}$ is the radius of maximal influence region (Determined by parameter mining).

Figure 4b depicts a sample grid search within a target event's maximal influence region. From each dot in this figure, the distances to actuators are different, resulting in different physical influences. For each dot, the search outputs the ratio of the target event's impact to the aggregated influence from other events (Line 18). Higher values of this ratio indicate the target event's impact on that location is greater than other events' joint influence. Thus, the sensor can be placed in any convenient location where this ratio is > 1.

4) Robust Physical Fingerprint Generation: There must be at least one sensor that can distinguish each event from others to prevent the evasion attacks against EVS. Thus, IoT service providers need to either install a new sensor or relocate an existing one for each vulnerable event to the identified placement regions. The first option allows keeping the existing sensors' locations the same and the second is a cost-effective method that only leverages the existing sensors. Yet, relocating a sensor may cause another event's fingerprint to become

vulnerable if the sensor distinguishes that event. To prevent this, we output which sensors can be relocated by checking if the sensor uniquely distinguishes an event. However, if the number of sensors in the smart home is low, it may not be feasible to ensure each event is uniquely detected by a sensor by only relocating the existing sensors. Here, there is a trade-off between the number of events protected against evasion and the number of deployed sensors. Therefore, in such cases, IoT service providers must install new sensors in the identified regions to protect all events against evasion attacks.

After the sensor addition/relocation, we update the EVS fingerprints to prevent evasion attacks. We first remove any fingerprints extracted from relocated sensors' readings as they change based on the new locations. We next revise the fingerprints to reflect the expected sensor readings for the added or relocated sensors. Similar to EVS fingerprint extraction, this requires collecting sensor readings when events occur. This can be performed by conducting experiments or using the automata that model each event's influence on sensor readings. The first option uses real-world data to learn new fingerprints, yet, requires additional experiments. The second does not require experiments but relies on automata correctness.

VI. Implementation

We implement our system's evadable physical fingerprint discovery and software patching components in Python. We use the Pandas library to develop the aggregation functions for rule, feature and sensor data aggregation. For evadable fingerprint discovery, we iteratively take each event's fingerprint and compare it with each subset of others to check whether it can be encapsulated or concealed due to their influence. For software patching, we generate new fingerprints with the underlying EVS using the aggregated sensor measurements.

We implement sensor location patching within Matlab. We have constructed 18 automata for 12 actuators that influence three physical channels using the Simulink toolbox. We use S-TaLiRo [6] to compute the robustness of LTL formulas required for robustness-guided parameter mining. Lastly, we implement the circular grid search by executing the automata with different distances from events (See Appendix B for the details).

EVS Implementations for Evaluation. We have studied six recent EVS [11], [12], [25], [34], [67], [68] by analyzing their physical event fingerprint extraction logic and the fingerprints they infer in their evaluation. We found that these systems are vulnerable to the evasion attacks introduced in Section IV. This is because they do not consider the complex physical relations between events and sensor readings such as indistinguishable and opposing influences from multiple events.

Among these EVS, we evaluate our system on two of them, (1) a rule-based approach, HAWatcher [34] and (2) an ML-based approach, Peeves [11]. We select these approaches for our evaluation since they outperform prior EVS with comprehensive physical fingerprints. They are also evaluated on various sensors and actuators in different smart home settings and shown to exhibit high accuracy (on average, 74% detection rate for Peeves and 98% precision with 94% recall for HAWatcher).

For physical event fingerprint extraction, HAWatcher leverages the Jenks algorithm [48] to first classify sensor readings as 'high' or 'low'. It then conducts hypothesis testing on the correlation between each event and classified sensor readings to generate rules as event fingerprints. Peeves constructs models using binary support vector machine (SVM) classifiers with a linear kernel as event fingerprints. It uses five features (min, max, mean, sum, and std) from a window of sensor readings. It leverages relative mutual information (RMI) to measure each feature's distinctiveness in distinguishing event-occurred and event-not-occurred classes. It uses the features above an RMI threshold (0.4) while learning event models.

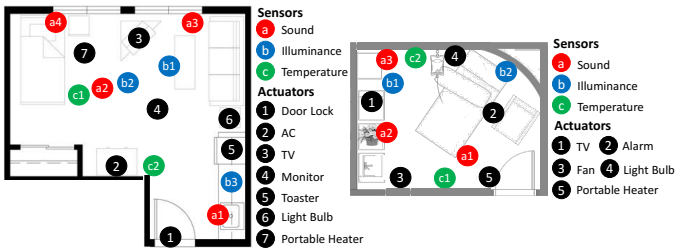


Fig. 5: Smart home settings in our evaluation: (Left) Studio apartment setup, (Right) patient care room setup.

We re-implemented these EVS. To verify our implementations are correct, we conducted data collection in our testbeds and extracted physical event fingerprints. We observed that influences on continuous channels (e.g., temperature) take a longer time to dissipate, making the usage of rules with high/low categories in HAWatcher infeasible. Thus, for continuous channels, we define the categories as 'Increase/Decrease' and extract the rules from the changes in sensor readings. We also found some event models of Peeves cannot be extracted in our testbeds due to the high RMI threshold (0.4). Thus, we also report our results when the RMI threshold is 0.2. After these minor modifications, HAWatcher and Peeves fingerprints detect all standard (non-evasion) spoofing and masking attacks with similar accuracy to their reported results (See Section VII-B).

VII. Evaluation

A. Evaluation Setup

We evaluate our system on two real-world smart home testbeds. The first is a studio apartment (Figure 5 (Left)), and the second is a patient care room (Figure 5 (Right)).

In the first testbed, we deployed 7 actuators and 9 sensors in a studio apartment. The sensors measure three physical channels (temperature, illuminance, and sound). We selected these sensor types as they are commonly deployed in smart homes for automation [20], [27], [56]. Specifically, we use photoresistor illuminance, BMP183 temperature, and LM393 sound detection sensors connected to Arduinos. The apartment includes widely deployed actuators such as a smart TV, light bulb, AC, and portable heater. We use a Raspberry Pi as the edge device to collect events and sensor readings. In this testbed, an author spent three days mimicking the events that sporadically occur during the apartment's regular use while ensuring each event occurs at least 10 times. We also triggered events with different actuator modes (e.g., heater with high/low and AC with different target temperature values). Thus, events are triggered in different conditions, and enough data is generated to learn fingerprints. We note that EVS use signal processing techniques

to eliminate the impact of environmental conditions and isolate the events' influence on sensor readings. This enables learning fingerprints from a few days of data [11].

In the second testbed, we deployed 5 actuators and 7 sensors in a room to convert it to a patient room. Particularly, we deployed TV, smart light bulb, and temperature regulation devices (heater and fan) that are used for stress reduction in hospital rooms [5]. We also installed an alarm for emergencies and sound, illuminance, and temperature sensors for monitoring [73]. Here, we use the same sensors with the studio apartment besides the sound sensor, which we replace with a numerical decibel-meter. We use different actuator devices from the studio except for the portable heater. In this testbed, two authors spent three days, one performing patient activities and the other performing visitor activities. Similar to the first testbed, to have enough coverage for learning event fingerprints, we ensure each event is triggered at least 10 times and with different actuator modes (e.g., fan with high/low settings).

In Appendix D, we detail the events in our testbeds and their influences on sensor readings. We then provide device parameters (e.g., sensor sensitivity levels) for both testbeds.

We contacted our university's IRB office and got advised that IRB approval is not required since our smart home environments are controlled testbeds, and we do not collect any sensitive information.

Research Questions. We present our results by focusing on several research questions:

RQ1 What are the physical event fingerprints that HAWatcher and Peeves extract?
RQ2 Which fingerprints are vulnerable to evasion attacks?
RQ3 How many attacks does EVS software patching prevent?
RQ4 How many attacks does sensor location patching prevent?
RQ5 How effective circular grid search is in determining the sensor's angular position?
RQ6 What is our system's performance overhead?

We run our evadable fingerprint discovery, software patching and sensor location patching algorithms on a laptop computer with a 1.4 GHz 4-core Intel i5 processor and 16 GB RAM.

B. Effectiveness

We first extract the physical event fingerprints in our testbeds through the physical fingerprint generation methods of HAWatcher and Peeves (Detailed in Section VI). We then identify the evadable fingerprints and show how attackers can evade these EVS. We apply software patching, and show that it prevents the $\texttt{Evasion}_1$ and $\texttt{Evasion}_2$ attacks. We next apply sensor location patching and show it prevents all remaining evasion attacks. We first detail our results with HAWatcher (Section VII-B1) and then with Peeves (Section VII-B2).

1) Evaluation Results with HAWatcher: HAWatcher extracts a total of 12 rules as event fingerprints, where nine of them are vulnerable to at least one evasion attack. Our software patching prevents the evasion attacks on seven of the nine rules. We prevent the remaining attacks with location patching, where relocating two sensors ensures all rules are robust to evasion.

TABLE I: Event rules before patching.

ID	Event Rule	Evasion		
		1	2	3
Studio Apartment Setup				
$\texttt{Rule}_1$	❶ ↔ {a1 = High}	✓	✗	✓
$\texttt{Rule}_2$	❷ ↔ {a2 = High, c1 = Dec, c2 = Dec}	✗	✓	✗
$\texttt{Rule}_3$	❸ ↔ {a2 = High, a3 = High, b1 = High, b2 = High}	✗	✗	✗
$\texttt{Rule}_4$	❹ ↔ {b1 = High, b2 = High}	✓	✗	✗
$\texttt{Rule}_5$	❺ ↔ {a1 = High}	✓	✗	✓
$\texttt{Rule}_6$	❻ ↔ {b1 = High, b2 = High, b3 = High}	✗	✗	✗
$\texttt{Rule}_7$	❼ ↔ {a4 = High, c1 = Inc}	✗	✓	✗
Patient Care Room Setup				
$\texttt{Rule}_8$	❶ ↔ {a2 = High, a3 = High, b1 = High}	✓	✗	✗
$\texttt{Rule}_9$	❷ ↔ {a1 = High, a2 = High, a3 = High}	✓	✗	✗
$\texttt{Rule}_{10}$	❸ ↔ {a1 = High, a2 = High, c1 = Dec}	✗	✓	✗
$\texttt{Rule}_{11}$	❹ ↔ {b1 = High, b2 = High}	✗	✗	✗
$\texttt{Rule}_{12}$	❺ ↔ {a1 = High, c1 = Inc}	✗	✓	✗

✓ means attack is successful, ✗ means it is unsuccessful.

(RQ1) Physical Event Fingerprints before Patching. Table I describes the event rules in the studio ($\texttt{Rule}_1$-$\texttt{Rule}_7$) and the patient room ($\texttt{Rule}_8$-$\texttt{Rule}_{12}$). For instance, ($\texttt{Rule}_1$) states the `a1` sound sensor must measure `High` when `door-unlock` occurs.

To test the correctness of the extracted rules, we evaluate their effectiveness in identifying standard (non-evasion) event spoofing and masking attacks. Similar to HAWatcher, we manipulate the event logs to conduct these attacks. We randomly inject 12 events (one of each event type) to the logs, while a physical event is not occurring, to conduct spoofing attacks. We remove one of each event from the logs to conduct masking attacks. We confirm that HAWatcher's rules detect all these attacks without any false positives or negatives.

(RQ2) Evasion Attacks. Our physical fingerprint analysis shows that $5/7$ (71%) of the rules in the studio apartment and $4/5$ (80%) of the rules in the patient care room are vulnerable to at least one evasion attack. Table I shows the vulnerable fingerprints, and we detail them below.

In the studio apartment, $\texttt{Evasion}_1$ (spoofing) attacks are successful on $3/7$ (43%) of the rules, and $\texttt{Evasion}_2$ (masking) and $\texttt{Evasion}_3$ (mask-and-spoof) attacks are successful on $2/7$ (29%) of them. $\texttt{Evasion}_1$ attacks occur if an event's rule is encapsulated by another rule. For instance, $\texttt{Rule}_1$ and $\texttt{Rule}_5$ are vulnerable to spoofing as both the `door-unlock` and `toaster-off` events only influence the `a1` sound sensor's readings where it outputs `sound-detected` when any of these events occur. Similarly, $\texttt{Evasion}_2$ attacks occur due to indistinguishable rules. Thus, rules extracted from `door-unlock` and `toaster-off` are vulnerable to mask-and-spoof attacks as well. $\texttt{Evasion}_3$ attacks occur due to concealing influences from events. $\texttt{Rule}_2$ and $\texttt{Rule}_7$ are vulnerable to $\texttt{Evasion}_3$ as `AC-on` and `heater-on` influence the `c1` temperature sensor in opposing ways.

In the patient room, $\texttt{Evasion}_1$ and $\texttt{Evasion}_2$ attacks are each successful on $2/5$ (40%) of the rules. More events influence the same sensor's readings as the physical space is smaller, resulting in a denser device deployment and causing aggregated evasion attack vulnerabilities. For instance, $\texttt{Rule}_9$ is encapsulated by the joint influence from `TV-on` and `fan-on`, creating an opportunity for spoofing `alarm-on` when these events occur.

(RQ3) Software Patching. Table II describes the aggregated event rules generated by software patching of HAWatcher. The aggregated event rules prevent $3/7$ (43%) of the evasion attacks in the studio and $4/4$ (100%) of the attacks in the patient room.

TABLE II: Aggregated event rules after software patching.

ID	Aggregated Event Rule	Evasion 1	2	3
Studio Apartment Setup				
$\texttt{Rule}_{13}$	❶ & ❺ ↔ {a1 = High}	✓	✗	✓
$\texttt{Rule}_{14}$	❸ & ❹ ↔ {a2 = High, a3 = High, b1 = Agg_High, b2 = Agg_High}	✗	✗	✗
$\texttt{Rule}_{15}$	❹ & ❻ ↔ {b1 = Agg_High, b2 = Agg_High, b3 = High}	✗	✗	✗
$\texttt{Rule}_{16}$	❷ & ❼ ↔ {a4 = High, c2 = Dec}	✗	✗	✗
Patient Care Room Setup				
$\texttt{Rule}_{17}$	❶ & ❷ & ❹ ↔ {a1 = High, a2 = Agg_High, a3 = Agg_High, b1 = Agg_High, b2 = High}	✗	✗	✗
$\texttt{Rule}_{18}$	❶ & ❷ & ❸ ↔ {a1 = Agg_High, a2 = Agg_High, a3 = Agg_High, b1 = High, c1 = Dec}	✗	✗	✗
$\texttt{Rule}_{19}$	❶ & ❷ & ❺ ↔ {a1 = Agg_High, a2 = Agg_High, a3 = Agg_High, b1 = High, c1 = Inc}	✗	✗	✗
$\texttt{Rule}_{20}$	❸ & ❺ ↔ {a1 = Agg_High, a2 = High}	✗	✗	✗

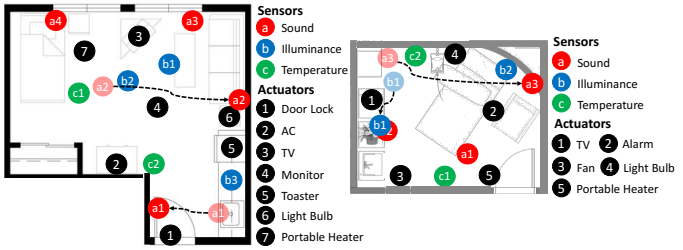


Fig. 6: Sensor placement after location patching in the (Left) studio apartment, (Right) patient care room. The dashed lines show how the sensors are moved from their initial placement.

The aggregated rules prevent the evasion attacks by dynamically adjusting the expected sensor readings based on the other events that occur. For instance, an aggregated rule in the patient room defines the aggregated influence from `alarm-on`, `TV-on` and `fan-on` ($\texttt{Rule}_{18}$). It prevents the $\texttt{Evasion}_1$ attack against the alarm because when `TV-on` and `fan-on` occur and the adversary spoofs `alarm-on`, HAWatcher checks $\texttt{Rule}_{18}$. Since the influence from `TV-on` and `fan-on` cannot satisfy $\texttt{Rule}_{18}$, HAWatcher detects this attack (Detailed in Section VII-C).

Software patching cannot prevent four evasion attacks against $\texttt{Rule}_1$ and $\texttt{Rule}_5$. These rules only rely on a single sensor (`a1`) that outputs boolean-typed readings, preventing the aggregated rules from distinguishing these events.

(RQ4) Sensor Location Patching. We prevent the four evasion attacks in the studio that software patching could not prevent by relocating `a1` and `a2` sound sensors, as depicted in Figure 6 (Left). When `a1` sensor is relocated, it only measures the door lock's influence as its distance with the toaster is increased, preventing the evasion attacks against `door-unlock`.

After this relocation, none of the sensors measure the toaster events, and therefore, sensor location patching outputs `a2` as a candidate to be relocated to measure its influence. Here, `a2` sensor is selected since all other sound sensors uniquely fingerprint another event, but `a2` does not. After relocating `a2`, we update its rule as ❺ ↔ `a2` = `High`, and remove `a2` from other rules because the actuator automata indicate the sound sensor cannot measure other events' influences in its new location. After relocating the sensors, we re-conducted data collection and confirmed the rules learned from automata outputs are correct, preventing the evasion attacks. Similarly, adding sensors to the identified locations prevents the evasion attacks.

2) Evaluation Results with Peeves: Peeves extracts a total of 12 models as event fingerprints, where eight of them are vulnerable to at least one evasion attack. Our software patching prevents the evasion attacks on $4/8$ (50%) of the models. Sensor location patching prevents the remaining attacks, where relocating four sensors ensures all models are robust.

(RQ1) Physical Event Fingerprints before Patching. We extract seven event models for the studio and five models for the patient room, with different SVM kernels (linear and RBF) and parameters (RMI threshold and window size). We observe some models cannot be extracted when the RMI threshold is 0.4, but all are extracted when it is 0.2. This is because when multiple events influence the same sensor's readings, the distinctiveness of the features extracted from them decreases.

To test the event models' accuracy, we conduct five-fold cross validation on our collected data. Table III presents the accuracy of event models with different sets of parameters. We observe all models yield high accuracy (> 0.73) and most have perfect accuracy. The accuracies of our models are similar to the results in [11], where the average detection rate is 0.74.

(RQ2) Evasion Attacks. When the RMI threshold is set as 0.2, our physical fingerprint analysis indicates that $4/7$ (57%) of the models in the studio and $4/5$ (80%) of them in the patient room are vulnerable to at least one evasion attack.

In the studio apartment, $\texttt{Evasion}_1$ (spoofing) and $\texttt{Evasion}_3$ (mask-and-spoof) attacks are successful on `door-unlock` and `toaster-off`, and $\texttt{Evasion}_2$ (masking) attacks are successful on `AC-on` and `heater-on`. Compared to HAWatcher, Peeves can successfully detect evasion attempts against `monitor-on` since numerical features enable distinguishing this event. We also noticed some models prevent spoofing and mask-and-spoof attacks against `door-unlock` and `toaster-off`. Upon further examination, we found their similar influence on `a1` sound sensor readings still disrupts the models, causing false positives (physically occurred event's notification is flagged as an attack).

In the patient room, we observe the same evasion attacks against HAWatcher are applicable to Peeves. This is because when multiple events occur, they influence sensor readings similarly with another event's influence, causing misclassifications. For instance, an adversary can spoof the `TV-on` event when `fan-on` and `alarm-on` occur as these events influence `a2` sound and `b1` illuminance sensors at similar levels with `TV-on`.

(RQ3) Software Patching. The software-patched event models detect $2/6$ (33%) of the evasion attacks in the studio and $2/4$ (50%) of the evasion attacks in the patient room.

In the studio apartment, we derive a model for `door-unlock` and `toaster-off`, and another for `AC-on` and `heater-on`. The first model cannot prevent the evasion attacks since the sound sensor that detects the `door-unlock` and `toaster-off` events output boolean-typed readings. The second model prevents the $\texttt{Evasion}_2$ attacks against `AC-on` and `heater-on` as their aggregated influence has a unique fingerprint on sensor readings.

In the patient room, we derive a joint model for `fan-on` and `heater-on`, which successfully prevents the $\texttt{Evasion}_2$ attacks. We derive three separate models to prevent the $\texttt{Evasion}_1$ attacks against `TV-on` and `alarm-on`. This is because their event models can be evaded when different combinations of other events physically occur. However, we notice that the new event models

TABLE III: Average accuracy of event models with five-fold cross validation and their vulnerability to evasion.

Event Model Parameters			Studio Apartment Setup										Patient Care Room Setup							
			Average Accuracy							Vulnerable Models			Average Accuracy					Vulnerable Models		
Kernel	RMI Threshold	Window Size	❶	❷	❸	❹	❺	❻	❼	Evasion_1	Evasion_2	Evasion_3	❶	❷	❸	❹	❺	Evasion_1	Evasion_2	Evasion_3
Linear	0.2	5sec/1min	0.77	0.8	1	0.88	0.74	1	1	❶, ❺	❷	❶, ❺	0.78	0.78	0.87	0.86	0.73	❶, ❷, ❸	❺	✗
		10sec/3min	0.77	1	0.89	0.88	0.76	1	1	❶, ❺	✗	❶, ❺	0.78	0.78	1	0.86	1	❶, ❷	❸, ❺	✗
		15sec/5min	0.77	1	0.89	0.88	0.74	1	1	❶, ❺	✗	❶, ❺	0.78	0.78	1	0.86	1	❶, ❷	❸, ❺	✗
	0.4	5sec/1min	0.74	–	1	–	–	–	1	❶	✗	✗	–	–	–	0.86	–	✗	✗	✗
		10sec/3min	–	1	1	–	–	0.86	1	✗	✗	✗	–	–	–	0.86	1	✗	❺	✗
		15sec/5min	–	1	1	–	–	–	1	✗	✗	✗	–	0.8	1	0.86	1	❷	❸, ❺	✗
RBF	0.2	5sec/1min	0.74	1	1	1	0.74	1	1	❶	❷	✗	1	1	0.94	1	0.73	❶, ❷, ❸	❺	✗
		10sec/3min	0.77	1	1	1	0.74	1	1	❶, ❺	✗	❶, ❺	1	1	1	1	1	❶, ❷	❸, ❺	✗
		15sec/5min	0.74	1	1	1	0.74	1	1	❺	❷, ❼	✗	1	1	1	0.86	1	❶, ❷	❸, ❺	✗
	0.4	5sec/1min	–	–	1	–	–	–	1	✗	✗	✗	–	–	–	1	–	✗	✗	✗
		10sec/3min	–	1	1	–	–	–	1	✗	✗	✗	–	–	–	1	0.88	✗	❺	✗
		15sec/5min	0.74	–	0.99	–	–	–	0.86	✗	❼	✗	–	–	0.89	1	0.88	✗	❸, ❺	✗

(–) indicates that event model could not be derived because none of the features were distinctive based on the RMI threshold

are also vulnerable to evasion since they cannot di
which subset of events has occurred. For instance, w
a model for TV-on, alarm-on and fan-on events, but i
events-occurred when TV-on, alarm-on and light-
This would allow an adversary to spoof alarm-on v
other events occur. Thus, we prevent the Evasion
against TV-on and alarm-on with sensor location patc

(RQ4) Sensor Location Patching. We prevent four attacks in the studio by relocating two sensors and two attacks in the patient care room by relocating two sensors.

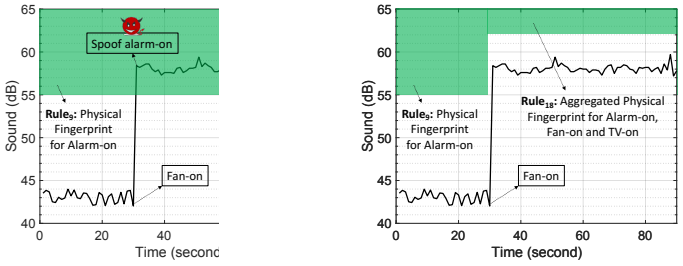
In the studio, we relocate the a1 and a2 sound sensors as depicted in Figure 6 (Left). This enables deriving unique models for door-unlock and toaster-off events, preventing the Evasion_1 and Evasion_3 attacks. We confirm the new models are not vulnerable to evasion by conducting physical fingerprint analysis again with the new models.

In the patient room, we relocate the b1 illuminance and a3 sound sensors as shown in Figure 6 (Right). After the relocations, the features extracted from b1 enables fingerprinting the TV-on event and features from a3 distinguishes the alarm-on event. Thus, the event models learned after the sensor location patching are not vulnerable to evasion. We confirmed this by first learning new event models using the traces generated from automata outputs. We next conducted data collection with the new sensor placement and tested if event models are vulnerable to evasion. Our experiments confirmed the new event models are not vulnerable to evasion (Detailed in Section VII-C).

C. Case Studies

We detail two evasion attacks, one against HAWatcher and another against Peeves, and then show how they are mitigated after software and sensor location patching. We also provide a case study to evaluate the effectiveness of circular grid search, which shows putting the sensors merely close to the actuators may not be enough to prevent the evasion attacks.

Case Study 1. We detail the spoofing attack against alarm-on (Rule_9) in the patient room. Rule_9 indicates alarm-on influences three sound sensors as $\{\text{a1} = \text{High}, \text{a2} = \text{High}, \text{a3} = \text{High}\}$. Yet, when fan-on and TV-on physically occur, these three sensors measure the expected High readings. For instance, Figure 7a depicts a1's readings before software patching. Fan-on's influence satisfies Rule_9's expected readings for a1. If the adversary spoofs the alarm-on event when fan-on and TV-on events occur, HAWatcher cannot detect this. Through this attack, the adversary can cause unnecessary panic at the patient room.



(a) The evasion attack ful before software patching.

(b) The evasion attack is mitigated after software patching.

Fig. 7: a1 sound sensor measurements in the patient care room. (a) The adversary can evade EVS by spoofing alarm-on when fan-on occurs. (b) Software patching prevents the attack.

After software patching, we define a new rule (Rule_{18}) from the aggregated influences from alarm-on, fan-on and TV-on. Rule_{18} defines that when these three events occur together, the expected sensor measurements are $\{\text{a1} = \text{Agg\_High}, \text{a2} = \text{Agg\_High}, \text{a3} = \text{Agg\_High}, \text{b1} = \text{High}, \text{c1} = \text{Dec}\}$. If the adversary spoofs the alarm-on event when others occur, HAWatcher detects this as Rule_{18} gets violated. For instance, Figure 7b shows a1 readings after software patching with the new rule. Particularly, Rule_{18} checks the aggregated influence from alarm-on event and fan-on when it receives their notification instead of checking their rules individually. Therefore, it detects the alarm-on spoofing attack as the influence level from fan-on is lower than the expected measurement.

Case Study 2. We detail the spoofing attack against Peeves's TV-on model in the patient room. Figure 8 illustrates the features extracted from a1 sound and b1 illuminance sensor readings before and after sensor location patching. Before patching, the features cannot distinguish the TV-on event's influence from the joint influence from alarm-on and light-on events. This is because the TV-on and alarm-on events influence the a1 sound sensor measurements, and TV-on and light-on influence b1 illuminance sensor measurements at similar levels.

After sensor location patching all features are able to distinguish TV-on from other events. For instance, the features extracted from the b1 illuminance sensor (right two columns

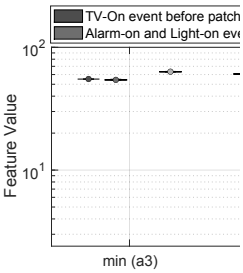
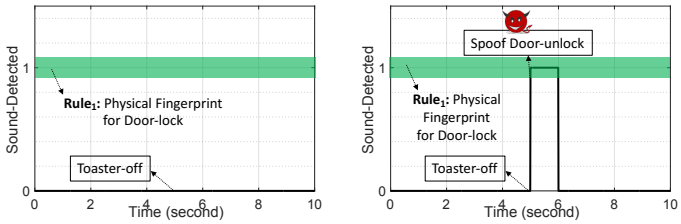


Fig. 8: Features (min, sensor measurements l



(a) The evasion attack is mitigated with the correct patch.

(b) The evasion attack is successful with the incorrect patch.

Fig. 9: a1 sound sensor measurements in the studio apartment when it is placed at the same distance to the door lock but at different angles ($\alpha = 90, \beta = 180$ vs $\alpha = 270, \beta = 180$).

for each feature) can fingerprint TV-on. This is because, with sensor relocation, b1's distance to the TV stays similar, but its distance to the light bulb increases. This decreases the light-on event's influence on the sensor's measurements.

Case Study 3 (RQ5: Circular Grid Search Effectiveness). One may put the sensors close to actuators to address the evasion attacks. Yet, besides distance, the angle where the sensor is located is also critical as it impacts the sensor's distance to other actuators. We conduct an additional experiment to show the importance of the sensor's angle to the actuator. For this, we place the a1 sound sensor on an angle that is not suggested by the circular grid search algorithm. Figure 9 illustrates the door-unlock event's rule ($\texttt{Rule}_1$) and sound sensor measurements when toaster-on occurs with the correct sensor location ($\alpha = 90, \beta = 180$) and an incorrect sensor location ($\alpha = 270, \beta = 180$) where the distances to the door lock are the same. Although $\texttt{Rule}_1$ does not change, toaster-off event does not influence the sound sensor with the correct location, while it creates an indistinguishable influence with the door-unlock event with the incorrect location. The incorrect location makes the rule evadable, enabling the adversary to conduct a door-unlock spoofing attack. Through this, the adversary can trigger the IoT apps conditioned on door-unlock, such as causing the studio apartment to enter the "home mode" and opening the patio door [11], [20].

D. (RQ6) Performance Evaluation

Parameter Mining. The time required for parameter mining depends on the automata. Instant influences are modeled with algebraic equations, and continuous influences are modeled with differential equations and take more time. Mining sound and illuminance distances take 1.51 ± 0.32 secs. For temperature, parameter mining takes 8.40 ± 3.33 secs. Parameter mining is repeated twice for each physical channel and the actuators impacting that channel. Thus, its total cost is 36.77 secs for the studio apartment and 35.96 secs for the patient room.

Circular Grid Search. Circular grid search is conducted with angle and distance step parameters for sensor placement. With different parameter selections, the efficiency of grid search changes linearly. We select these parameters to be $\alpha = 45$, $\beta = 45$ and $\mathtt{s} = 0.1$. Circular grid search takes 220.71 ± 53.01 secs for sound and illuminance, and 751.26 ± 139.42 secs for temperature. This search is repeated for any physical channel–event pair if parameter mining cannot determine a clear placement region. In our experiments, seven pairs require circular grid search, with a total cost of 44 mins.

VIII. Limitations and Discussion

Deploying Our Defenses in Different Smart Homes. EVS may require generating new event fingerprints in other smart homes since the types, locations, and vendors of devices may differ from each other. Thus, after EVS generate fingerprints in a smart home, our defenses are then used to detect evadable fingerprints and protect them against evasion attacks.

Our evadable fingerprint identification and software patching are automated processes that create new fingerprints using existing data collected for EVS fingerprint generation. Sensor location patching constructs automata to model an event's physical influence(s). We define generic flow functions that can be used in other smart homes for the studied physical channels, as we detail in Appendix C, but there could be sensors measuring different channels. In such cases, a new flow function can be either (1) provided by device manufacturers or (2) defined from control theory [42], [56], [74].

Our defenses then require executing the automata with three parameters specific to each smart home's devices for parameter mining and circular grid search. First, to set the distance parameter, we use the Lumos localization tool [65], which requires a user to walk inside the smart home with their mobile phone or tablet. Lumos accurately identifies device locations in 30 mins, consisting of 27 mins of wireless sniffing followed by three mins of walking. Second, the min/max output parameters are set based on the highest and lowest possible readings of the installed sensors. Lastly, as detailed in Section V-C, the device property parameter is automatically determined using existing EVS event and sensor measurement traces. Sensor location patching then outputs placement regions, where IoT service providers must relocate or add a minimal number of sensors.

Concept Drift. The relationship between events and sensor readings may dynamically change over time (e.g., due to environmental noise and seasonal changes), creating a concept drift [35] in event fingerprints. This could cause false negatives (missed attacks) and false positives (falsely flagged attacks) in EVS over time. To address this, EVS conduct differencing to remove the impact of other factors on sensors and isolate the events' influences. Yet, in rare cases, concept drift may occur in EVS (e.g., due to device aging [60]). To handle them, EVS can integrate sophisticated concept drift detection mechanisms [80].

Fully Overlapped Events. There may be rare cases where actuators are placed very closely, and an event's influence on a sensor's measurements is always lower than other events. In

these cases, location patching may not identify a placement region for that event-sensor pair. Thus, such events must be distinguished through sensors with other modalities.

Number of Detected Events. To prevent evasion attacks, sensor location patching ensures at least one sensor can uniquely distinguish an event's influence by minimizing the influences from other events on the sensor's measurements. This may result in a sensor being able to detect a single event, yet, a user may desire their sensor to measure multiple events. To ensure a sensor detects the maximum number of events while preventing evasion attacks, IoT service providers can prefer the sensor placement regions where multiple events' influences overlap, yet, the target event's influence is the highest.

Immovable Sensors. In smart homes, there could be sensors attached to a specific actuator (e.g., door or window contact sensors), which are infeasible to move to other locations. Using our location patching for these sensors is not practical. However, these sensors can already distinguish the events of the actuator they are attached to, and therefore, they can prevent evasion attacks against these events' fingerprints.

IX. Related Work

Event Verification Systems. There has been an increasing number of works in EVS, and more broadly anomaly detection in smart homes [11], [12], [34], [67], [68], [81]. These systems learn physical fingerprints of events on sensor measurements offline. They then detect attacks if the sensor readings deviate from the learned physical behavior of an event at runtime. Unfortunately, as we have shown in Section VI, these efforts have not considered the complex physical properties that enable evasion attacks in their design, which motivates our approach. Our system is complementary to these EVS, where we introduce EVS software patching and sensor location patching to make them robust against identified evasion attacks.

Physics-based Attack Detection (PBAD). Cyber-Physical Systems (CPS) are broadly vulnerable to three types of attacks [37]. First, an adversary can conduct control command attacks, which cause actuators to execute commands different from the controller's intended action, by compromising actuators or injecting malicious commands. Second, an adversary can inject false sensor readings to deceive the controller about the real state of the CPS. Lastly, an adversary can compromise the CPS controller to issue malicious commands. Event spoofing and masking attacks in smart homes are a subset of the control command attacks in CPS, where an adversary causes discrepancies between the cyber and physical actuator states.

To detect these attacks, PBAD has received attention from both security and control theory communities [37] in various CPS domains such as control systems [36], [41], [51], [52], [72], water distribution systems [7], [32], chemical plants [16], [54], and autonomous vehicles [24], [26], [59], [66]. These systems learn the time-series model of a CPS that predicts future sensor readings from current sensor readings and control commands using subspace model identification. They then compute residuals, which are the deviations of the real sensor readings from the predicted readings. They detect an attack if the residual exceeds a threshold at any single time or the cumulative sum of the historical residuals exceeds a threshold.

The main difference between EVS and PBAD systems is how they generate fingerprints or physical models for attack detection. PBAD systems leverage state-space models to model the closed-loop control of CPS. This is because CPS includes a sophisticated closed-loop control that continuously determines actuator signals based on sensor readings. For instance, robotic vehicles continuously issue torque and thrust values to rotors to keep the roll, pitch, and yaw angles at target values [59]. In contrast, EVS fingerprints simply define each event's influence directly on the sensor readings. Although certain smart home actuators may have a closed-loop control (e.g., heater and AC), users can also turn on these devices manually and using smartphone companion apps. For example, an IoT app may automatically turn on a heater when the temperature is `65°F` but the user may also manually turn it on when it is `70°F`. Thus, it is impractical to use PBAD state-space models in smart homes due to the unpredictable nature of smart home events.

There have also been stealthy attacks proposed against CPS, which create small deviations in sensor and actuator signals over time to cause maximum damage without being detected by PBAD systems [59], [72]. Such stealthy attacks are similar to our attacks since they also evade defense mechanisms. However, unlike our evasion attacks, stealthy attacks inject slight deviations indistinguishable from the natural perturbations of a CPS to evade PBAD. In this paper, we focus on the evasion attacks against EVS due to the events' encapsulating, concealing, and indistinguishable influences on sensor readings and propose two complementary defenses against these attacks.

Evasion Attacks. Evasion attacks bypass security defense mechanisms to exploit vulnerabilities in a target system without detection. Previous works have demonstrated evasion attacks against security applications (oftentimes built on ML and DNN models), such as intrusion detection [33], [43], malware detection [10], [69], [79] and autonomous systems [15], [23], [46], [57]. A line of work has explored evasion attacks against anomaly detection systems for CPS, e.g., water treatment plants [30], and water distribution systems [29], [50].

These works induce model behavior by perturbing inputs of systems (e.g., spoofing sensor values, injecting commands, perturbing malware packets, and binaries) digitally or physically to control the model output chosen by the adversary. In contrast, we target a separate class of evasion attacks on EVS fingerprints by exploiting complex physical relations between events and sensor readings in smart homes. We also propose two defenses to make smart homes robust against these attacks.

Sensor Placement. Prior works have studied sensor placement to detect air contamination in buildings [28], and to optimize energy usage in hospitals [55], offices [82], data centers [76], and smart parking [9]. Others have studied sensor placement for energy harvesting [14], battery efficiency in wireless sensor networks [21], [47], CPS monitoring [49], and fault detection [62], [78]. These works find a sensor placement for a specific objective; thus, they cannot be extended for EVS, which require a different analysis on the events' influences on sensor readings based on their fingerprints. To the best of our knowledge, we introduce the first work that formalizes sensor placement to prevent evasion attacks against EVS.

X. Conclusion

In this paper, we show that EVS are vulnerable to evasion attacks in which the attacker leverages the complex physical

properties between events and sensors to conduct event spoofing and masking attacks without getting detected. To address this, we propose software and sensor location patching as a defense against the evasion attacks on EVS. The evaluation of our prototype on one rule-based and one ML-based EVS deployed in two smart home settings shows that our system can effectively detect and prevent the evasion attacks.

ACKNOWLEDGMENT

This work has been partially supported by the National Science Foundation (NSF) under grant CNS-2144645 and Office of Naval Research (ONR) under grant N00014-20-1-2128. The views expressed are those of the authors only.

REFERENCES

[1] H. Abbas, H. Mittelmann, and G. Fainekos, "Formal property verification in a conformance testing framework," in *ACM/IEEE Conference on Formal Methods and Models for Codesign (MEMOCODE)*, 2014.

[2] A. Acar, H. Fereidooni, T. Abera, A. K. Sikder, M. Miettinen, H. Aksu, M. Conti, A.-R. Sadeghi, and S. Uluagac, "Peek-a-boo: I see your smart home activities, even encrypted!" in *ACM Conference on Security and Privacy in Wireless and Mobile Networks*, 2020.

[3] M. Ahrens, "Home smoke alarms: the data as context for decision," *Fire Technology*, 2008.

[4] C. Alcaraz and J. Lopez, "Wide-area situational awareness for critical infrastructure protection," *IEEE Computer*, 2013.

[5] C. C. Andrade and A. S. Devlin, "Stress reduction in the hospital room: Applying ulrich's theory of supportive design," *Journal of environmental psychology*, 2015.

[6] Y. Annpureddy, C. Liu, G. Fainekos, and S. Sankaranarayanan, "S-taliro: A tool for temporal logic falsification for hybrid systems," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2011.

[7] W. Aoudi, M. Iturbe, and M. Almgren, "Truth will out: Departure-based process-level detection of stealthy attacks on control systems," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2018.

[8] N. Apthorpe, D. Reisman, S. Sundaresan, A. Narayanan, and N. Feamster, "Spying on the smart home: Privacy attacks and defenses on encrypted IoT traffic," *arXiv preprint arXiv:1708.05044*, 2017.

[9] A. Bagula, L. Castelli, and M. Zennaro, "On the design of smart parking networks in the smart cities: An optimal sensor placement model," *Sensors*, 2015.

[10] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrndić, P. Laskov, G. Giacinto, and F. Roli, "Evasion attacks against machine learning at test time," in *European conference on machine learning and knowledge discovery in databases*, 2013.

[11] S. Birnbach, S. Eberz, and I. Martinovic, "Peeves: Physical event verification in smart homes," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2019.

[12] S. Birnbach, S. Eberz, and I. Martinovic, "Haunted house: physical smart home event verification in the presence of compromised sensors," *ACM Transactions on Internet of Things*, 2021.

[13] J. Boorman, B. Green, and D. Prince, "Transformers: Intrusion detection data in disguise," in *Computer Security*, 2020.

[14] O. M. Bushnaq, A. Chaaban, S. P. Chepuri, G. Leus, and T. Y. Al-Naffouri, "Sensor placement and resource allocation for energy harvesting IoT networks," *Digital Signal Processing*, 2020.

[15] X. Cao and N. Z. Gong, "Mitigating evasion attacks to deep neural networks via region-based classification," in *Annual Computer Security Applications Conference (ACSAC)*, 2017.

[16] A. A. Cardenas, S. Amin, Z.-S. Lin, Y.-L. Huang, C.-Y. Huang, and S. Sastry, "Attacks against process control systems: risk assessment, detection, and response," in *ACM symposium on Information, Computer and Communications Security*, 2011.

[17] Z. B. Celik, E. Fernandes, E. Pauley, G. Tan, and P. McDaniel, "Program analysis of commodity IoT applications for security and privacy: Challenges and opportunities," *ACM Computing Surveys (CSUR)*, 2019.

[18] Z. B. Celik, P. McDaniel, and G. Tan, "Soteria: Automated IoT safety and security analysis," in *USENIX Annual Technical Conference (USENIX ATC)*, 2018.

[19] Z. B. Celik, P. McDaniel, G. Tan, L. Babun, and A. S. Uluagac, "Verifying internet of things safety and security in physical spaces," *IEEE Security & Privacy*, 2019.

[20] Z. B. Celik, G. Tan, and P. D. McDaniel, "IoTGuard: Dynamic enforcement of security and safety policy in commodity IoT." in *NDSS*, 2019.

[21] Y. Chen, C.-N. Chuah, and Q. Zhao, "Sensor placement for maximizing lifetime per unit cost in wireless sensor networks," in *IEEE Military Communications Conference (MILCOM)*, 2005.

[22] L. Cheng, K. Tian, and D. Yao, "Orpheus: Enforcing cyber-physical execution semantics to defend against data-oriented attacks," in *Annual Computer Security Applications Conference (ACSAC)*, 2017.

[23] A. Chernikova, A. Oprea, C. Nita-Rotaru, and B. Kim, "Are self-driving cars secure? evasion attacks against deep neural networks for steering angle prediction," in *IEEE Security and Privacy Workshops (SPW)*, 2019.

[24] H. Choi, W.-C. Lee, Y. Aafer, F. Fei, Z. Tu, X. Zhang, D. Xu, and X. Deng, "Detecting attacks against robotic vehicles: A control invariant approach," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2018.

[25] J. Choi, H. Jeoung, J. Kim, Y. Ko, W. Jung, H. Kim, and J. Kim, "Detecting and identifying faulty IoT devices in smart home with context extraction," in *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2018.

[26] P. Dash, G. Li, Z. Chen, M. Karimibiuki, and K. Pattabiraman, "Pid-piper: Recovering robotic vehicles from physical attacks," in *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2021.

[27] W. Ding, H. Hu, and L. Cheng, "IoTSafe: Enforcing safety and security policy with real IoT physical interaction discovery," in *NDSS*, 2021.

[28] D. G. Eliades, M. P. Michaelides, C. G. Panayiotou, and M. M. Polycarpou, "Security-oriented sensor placement in intelligent buildings," *Building and Environment*, 2013.

[29] A. Erba, R. Taormina, S. Galelli, M. Pogliani, M. Carminati, S. Zanero, and N. O. Tippenhauer, "Constrained concealment attacks against reconstruction-based anomaly detectors in industrial control systems," in *Annual Computer Security Applications Conference (ACSAC)*, 2020.

[30] A. Erba and N. O. Tippenhauer, "No need to know physics: Resilience of process-based model-free anomaly detection for industrial control systems," *arXiv preprint arXiv:2012.03586*, 2020.

[31] E. Fernandes, J. Jung, and A. Prakash, "Security analysis of emerging smart home applications," in *IEEE Symposium on Security and Privacy (S&P)*, 2016.

[32] L. Fillatre, I. Nikiforov *et al.*, "A statistical method for detecting cyber/physical attacks on scada systems," in *IEEE Conference on Control Applications (CCA)*, 2014.

[33] P. Fogla and W. Lee, "Evading network anomaly detection systems: formal reasoning and practical techniques," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2006.

[34] C. Fu, Q. Zeng, and X. Du, "Hawatcher: Semantics-aware anomaly detection for appified smart homes," in *USENIX Security*, 2021.

[35] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM Computing Surveys (CSUR)*, 2014.

[36] J. Giraldo, S. H. Kafash, J. Ruths, and A. A. Cardenas, "Daria: Designing actuators to resist arbitrary attacks against cyber-physical systems," in *IEEE European Symposium on Security and Privacy (Euro S&P)*, 2020.

[37] J. Giraldo, D. Urbina, A. Cardenas, J. Valente, M. Faisal, J. Ruths, N. O. Tippenhauer, H. Sandberg, and R. Candell, "A survey of physics-based attack detection in cyber-physical systems," *ACM Computing Surveys (CSUR)*, 2018.

[38] F. Goksel, M. O. Ozmen, M. Reeves, B. Shivakumar, and Z. B. Celik, "On the safety implications of misordered events and commands in IoT systems," in *IEEE Security and Privacy Workshops (SPW)*, 2021.

[39] "Grid dynamics - anomaly detection platform for IoT," https://www.griddynamics.com/solutions/anomaly-detection-industry-4-0, 2020, [Online; accessed 15-Apr-2022].

[40] T. Gu, Z. Fang, A. Abhishek, H. Fu, P. Hu, and P. Mohapatra, "IoTgaze: IoT security enforcement via wireless context analysis," in *IEEE Conference on Computer Communications*, 2020.

[41] D. Hadvziosmanovic, R. Sommer, E. Zambon, and P. H. Hartel, "Through the eye of the PLC: semantic security monitoring for industrial processes," in *Annual Computer Security Applications Conference (ACSAC)*, 2014.

[42] M. J. Hancock, "The 1-d heat equation," *MIT OpenCourseWare.*, 2006.

[43] M. Handley, V. Paxson, and C. Kreibich, "Network intrusion detection: Evasion, traffic normalization, and End-to-End protocol semantics," in *USENIX Security*, 2001.

[44] T. A. Henzinger, "The theory of hybrid automata," in *Verification of digital and hybrid systems*. Springer, 2000.

[45] B. Hoxha, A. Dokhanchi, and G. Fainekos, "Mining parametric temporal logic properties in model-based design for cyber-physical systems," *International Journal on Software Tools for Technology Transfer*, 2018.

[46] L. Huang, A. D. Joseph, B. Nelson, B. I. Rubinstein, and J. D. Tygar, "Adversarial machine learning," in *ACM Workshop on Security and Artificial Intelligence*, 2011.

[47] E. Jain and Q. Liang, "Sensor placement and lifetime of wireless sensor networks: theory and performance analysis," in *IEEE Global Telecommunications Conference (GLOBECOM)*, 2005.

[48] G. F. Jenks, "The data model concept in statistical mapping," *International yearbook of cartography*, 1967.

[49] J.-A. Jiang, J.-C. Wang, H.-S. Wu, C.-H. Lee, C.-Y. Chou, L.-C. Wu, and Y.-C. Yang, "A novel sensor placement strategy for an IoT-based power grid monitoring system," *IEEE Internet of Things Journal*, 2020.

[50] M. Kravchik and A. Shabtai, "Efficient cyber attack detection in industrial control systems using lightweight neural networks and pca," *IEEE Transactions on Dependable and Secure Computing*, 2021.

[51] M. Krotofil, J. Larsen, and D. Gollmann, "The process matters: Ensuring data veracity in cyber-physical systems," in *ACM Symposium on Information, Computer and Communications Security*, 2015.

[52] R. Lanotte, M. Merro, A. Munteanu, and L. Viganò, "A formal approach to physics-based attacks in cyber-physical systems," *ACM Transactions on Privacy and Security (TOPS)*, 2020.

[53] F. Mitschke, "Decibel units," in *Fiber Optics*, 2009.

[54] Y. Mo, R. Chabukswar, and B. Sinopoli, "Detecting integrity attacks on scada systems," *IEEE Transactions on Control Systems Technology*, 2013.

[55] E. Mousavi, A. Khademi, and K. Taaffe, "Optimal sensor placement in a hospital operating room," *IISE Transactions on Healthcare Systems Engineering*, 2020.

[56] M. O. Ozmen, X. Li, A. Chu, Z. B. Celik, B. Hoxha, and X. Zhang, "Discovering IoT physical channel vulnerabilities," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2022.

[57] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in *Asia Conference on Computer and Communications Security*, 2017.

[58] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, "Context aware computing for the internet of things: A survey," *IEEE Communications Surveys & Tutorials*, 2013.

[59] R. Quinonez, J. Giraldo, L. Salazar, and E. Bauman, "Savior: Securing autonomous vehicles with robust physical invariants," in *USENIX Security*, 2020.

[60] P. Refregier, *Noise theory and application to physics: from fluctuations to information*. Springer Science & Business Media, 2004.

[61] A. Salazar, "On thermal diffusivity," *European Journal of Physics*, 2003.

[62] R. Sarrate, V. Puig, T. Escobet, and A. Rosich, "Optimal sensor placement for model-based fault detection and isolation," in *IEEE Conference on Decision and Control*, 2007.

[63] "Shield IoT - secure your mass scale IoT networks," https://shieldiot.io, 2020, [Online; accessed 15-Apr-2022].

[64] O. B. Sezer, E. Dogdu, and A. M. Ozbayoglu, "Context-aware computing, learning, and big data in internet of things: a survey," *IEEE Internet of Things Journal*, 2017.

[65] R. A. Sharma, E. Soltanaghaei, A. Rowe, and V. Sekar, "Lumos: Identifying and localizing diverse hidden IoT devices in an unfamiliar environment," in *USENIX Security*, 2022.

[66] Y. Shoukry, P. Martin, Y. Yona, S. Diggavi, and M. Srivastava, "Pycra: Physical challenge-response authentication for active sensors under spoofing attacks," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2015.

[67] A. K. Sikder, L. Babun, H. Aksu, and A. S. Uluagac, "Aegis: a context-aware security framework for smart home systems," in *Annual Computer Security Applications Conference (ACSAC)*, 2019.

[68] A. K. Sikder, L. Babun, and A. S. Uluagac, "Aegis+ a context-aware platform-independent security framework for smart home systems," *Digital Threats: Research and Practice*, 2021.

[69] N. Srndic and P. Laskov, "Practical evasion of a learning-based classifier: A case study," in *IEEE Symposium on Security and Privacy (S&P)*, 2014.

[70] "Technaura," https://www.technaura.com/anomaly-detection, 2020, [Online; accessed 15-Apr-2022].

[71] R. Trimananda, J. Varmarken, A. Markopoulou, and B. Demsky, "Packet-level signatures for smart home devices," in *NDSS*, 2020.

[72] D. I. Urbina, J. A. Giraldo, A. A. Cardenas, N. O. Tippenhauer, J. Valente, M. Faisal, J. Ruths, R. Candell, and H. Sandberg, "Limiting the impact of stealthy attacks on industrial control systems," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2016.

[73] L. P. Voigt, K. Reynolds, M. Mehryar, W. S. Chan, N. Kostelecky, S. M. Pastores, and N. A. Halpern, "Monitoring sound and light continuously in an intensive care unit patient room: A pilot study," *Journal of Critical Care*, 2017.

[74] N. Voudoukis and S. Oikonomidis, "Inverse square law for light and radiation: A unifying educational approach," *European Journal of Engineering Research and Science*, 2017.

[75] Q. Wang, W. U. Hassan, A. Bates, and C. Gunter, "Fear and logging in the internet of things," in *NDSS*, 2018.

[76] X. Wang, X. Wang, G. Xing, J. Chen, C.-X. Lin, and Y. Chen, "Towards optimal sensor placement for hot server detection in data centers," in *IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2011.

[77] E. Winer, *The audio expert: everything you need to know about audio*. CRC Press, 2012.

[78] K. Worden and A. Burrows, "Optimal sensor placement for fault detection," *Engineering Structures*, 2001.

[79] W. Xu, Y. Qi, and D. Evans, "Automatically evading classifiers," in *NDSS*, 2016.

[80] L. Yang, W. Guo, Q. Hao, A. Ciptadi, A. Ahmadzadeh, X. Xing, and G. Wang, "CADE: Detecting and explaining concept drift samples for security applications," in *USENIX Security*, 2021.

[81] R. Yasaei, F. Hernandez, and M. A. Al Faruque, "IoT-CAD: context-aware adaptive anomaly detection in IoT systems through sensor association," in *IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2020.

[82] D. Yoganathan, S. Kondepudi, B. Kalluri, and S. Manthapuri, "Optimal sensor placement strategy for office buildings using clustering algorithms," *Energy and Buildings*, 2018.

[83] B. Yuan, Y. Jia, L. Xing, D. Zhao, X. Wang, and Y. Zhang, "Shattered chain of trust: Understanding security risks in cross-cloud IoT access delegation," in *USENIX Security*, 2020.

[84] B. Yuan, Y. Wu, M. Yang, L. Xing, X. Wang, D. Zou, and H. Jin, "Smartpatch: Verifying the authenticity of the trigger-event in the IoT platform," *IEEE Transactions on Dependable and Secure Computing*, 2022.

[85] W. Zhang, Y. Meng, Y. Liu, X. Zhang, Y. Zhang, and H. Zhu, "Homonit: Monitoring smart home apps from encrypted traffic," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2018.

[86] A. Zhivov, H. Skistad, E. Mundt, V. Posokhin, M. Ratcliff, E. Shilkrot, and A. Strongin, "Principles of air and contaminant movement inside and around buildings," in *Industrial ventilation design guidebook*, 2001.

[87] W. Zhou, Y. Jia, Y. Yao, L. Zhu, L. Guan, Y. Mao, P. Liu, and Y. Zhang, "Discovering and understanding the security hazards in the interactions between IoT devices, mobile apps, and clouds on smart home platforms," in *USENIX Security*, 2019.

APPENDIX A
APPENDIX GUIDE

In this appendix, we provide the information necessary to reproduce our results. Appendix B presents the implementation details of our sensor location patching. Appendix C presents the equations used for actuator automata. Appendix D presents the sensor and automata parameters.

APPENDIX B
IMPLEMENTATION DETAILS

Automata Construction. To implement our sensor location patching, we have constructed 18 automata for 12 actuators that influence three physical channels (temperature, illuminance and sound) to evaluate our system (See Figure 5). The sensors' modalities include both instant (illuminance and sound) and continuous (temperature) channels to show the effectiveness of our system for both types. We deploy sensors with boolean-typed attributes (e.g., `sound-detected`) and numerical readings (e.g., $\texttt{sound} = 60\ \texttt{dB}$). We constructed the automata and determined their parameters for all physical channels each event influences (Detailed in Appendix C and Appendix D).

Parameter Mining and Circular Grid Search. We implement the sensor location patching within the Matlab environment, since it enables executing the constructed automata with specific parameters and collecting data traces. We extend the S-TALIRO toolbox [6] to implement parameter mining. Particularly, we use DP-TALIRO, a dynamic programming-based algorithm, to compute the robustness of LTL formulas. The robustness of our LTL formulas monotonically changes with the changes in distance from the actuators. Thus, we construct a binary search-based algorithm that integrates DP-TALIRO as a subroutine to conduct robustness-guided parameter mining. Lastly, we implement the circular grid search by executing the automata with different distances from actuators determined by three search parameters. We set $\alpha = \beta = 45$ and $\mathtt{s} = 0.1$ so that a fine-grained search is performed with reasonable performance.

Peeves Implementation. Peeves uses five features, min, max, mean, sum and standard deviation, from a window of sensor measurements. To account for environmental noise, Peeves divides the sensor measurements into windows and subtracts the previous window's sensor value average from the current window's sensor measurements to isolate an event's influence on sensors. We set the time window as $5, 10, 15$ secs for instant physical channels and $1, 3, 5$ mins for continuous physical channels as it takes more time for events to influence sensor measurements. Peeves then leverages relative mutual information (RMI) to measure the distinctiveness of each feature in identifying whether an event occurred or not. If a feature's RMI is above a threshold (`th`), Peeves uses the feature in learning an event model. We set the RMI threshold as 0.2 and 0.4 to limit the noisy features while ensuring enough features can be extracted to learn event models. Peeves constructs event models as binary linear support vector machine (SVM) classifiers for each event. We also implement an SVM classifier with a radial basis function (RBF) kernel to compare its vulnerability to evasion with the linear kernel.

APPENDIX C
ACTUATOR AUTOMATA FLOW FUNCTIONS

We leverage the physical channels studied in control theory to determine the flow functions of the automata [42], [56], [74], [86]. The flow functions mathematically quantify how the physical channels (temperature, illuminance and sound) diffuse over air to reach different distances for the sensors to measure. Here, we present the equations used in our actuator automata. We note that more sophisticated flow functions that consider complex physical properties such as light and sound reflections can be easily integrated to our automata.

Temperature. Temperature is modeled with the heat diffusion equation [42], which is a partial differential equation that captures how heat dissipates through air from a point source.

$$\frac{\partial \mathtt{T}}{\partial \mathtt{t}} = \alpha \frac{\partial^2 \mathtt{T}}{\partial \mathtt{x}^2} \tag{1}$$

$$\mathtt{T}(\mathtt{e}, 0) = \mathtt{T_0} \tag{2}$$

$$\mathtt{T}(0, 0) = \mathtt{T_s} \tag{3}$$

In the equation, $\mathtt{T}$ is temperature in $^{\circ}\mathtt{K}$, $\mathtt{x}$ is the distance from the heat source in meters, α is the thermal diffusivity constant (in $\mathtt{m}^2/\mathtt{s}$), $\mathtt{e}$ is the maximum distance from the source, and $\mathtt{T_s}$ is the temperature of the source. We set the thermal diffusivity as $2.2 \cdot 10^{-5}\ \mathtt{m}^2/\mathtt{s}$ as a constant [61].

We also consider the airflow from `fan`, `heater` and `AC` that fastens the dissipation rate of the heat. Therefore, we multiply the thermal diffusivity (α) with a constant `c`. We set $\mathtt{c} = 100$ for the airflow from all three actuators.

Illuminance. Illuminance dissipates instantly over the air (at the speed of light), and therefore, modeled with an algebraic equation that relies on the inverse square law [74].

$$\mathtt{I_x} = \frac{\mathtt{I_s}}{4 \times \pi \times \mathtt{x}^2} \tag{4}$$

Here, $\mathtt{I_s}$ is the source's light intensity (in `lumens`), $\mathtt{x}$ is the distance from the source (in `meters`) and $\mathtt{I_x}$ denotes the luminosity flux at distance $\mathtt{x}$ (in `lux`).

Sound. Sound also dissipates instantly over the air so it is also modeled with an algebraic equation. The sound pressure level (`SP`) in decibels (`dB`) is defined as follows [77].

$$\mathtt{SP_2} = \mathtt{SP_1} + 20 \times \mathtt{log}_{10}(\frac{\mathtt{x_1}}{\mathtt{x_2}}) \tag{5}$$

In the formula, $\mathtt{SP_1}$ is the sound pressure level at distance $\mathtt{x_1}$ and $\mathtt{SP_2}$ is the sound pressure level at distance $\mathtt{x_2}$. Given $\mathtt{SP_1}, \mathtt{x_1}$ the sound pressure levels at any distance ($\mathtt{x_2}$) can be derived with this formula. As a standard, $\mathtt{x_1} = 1\ \mathtt{meters}$.

APPENDIX D
EVALUATION PARAMETERS

We present the evaluated events in our testbeds, and their influences on sensor measurements in Table IV.

TABLE IV: Events in our testbeds and sensors they influence.

Event	Sound Sensor	Illuminance Sensor	Temperature Sensor
Studio Apartment Setup			
Door-unlock	✓	✗	✗
AC-on	✓	✗	✓
TV-on	✓	✓	✗
Monitor-on	✗	✓	✗
Toaster-off	✓	✗	✗
Light-Bulb-on	✗	✓	✗
Heater-on	✓	✗	✓
Patient Care Room Setup			
TV-on	✓	✓	✗
Alarm-on	✗	✓	✗
Fan-on	✓	✗	✓
Light-Bulb-on	✗	✓	✗
Heater-on	✓	✗	✓

✓ means the event influences the sensor readings, ✗ means it does not.

TABLE V: Actuator automata parameters.

Actuator	Sound (dB)	Illuminance (lumens)	Temperature† (°F)
Studio Apartment Setup			
Door	57	n/a	n/a
AC	62	n/a	45
TV	58	250	n/a
Monitor	n/a	150	n/a
Toaster	60	n/a	n/a
Bulb	n/a	1200	n/a
Heater	52	n/a	95
Patient Care Room Setup			
TV	55	200	n/a
Alarm	65	n/a	n/a
Fan	61	n/a	60
Bulb	n/a	1000	n/a
Heater	52	n/a	95

† Temperature parameter denotes the temperature of the air directly exiting the actuator (not the target air temperature).

The actuators used in our two experiments are different from each other, except for the `TV`, `bulb` and `heater`. We use separate `TVs` and `bulbs` in the two environmental setups, but the same `heater`. We set the device property parameters in our actuator automata using the (τ, ϵ)-closeness metric, as described in Section V-C1. The details of the actuator parameters are presented in Table V.

We use a photoresistor illuminance sensor, a BMP183 temperature sensor, and an LM393 sound detection sensor in the studio apartment experiments. In the patient care room setup, we replace the sound sensor with a numerical decibel-meter. The threshold values used in dynamic testing and requirements identification (LTL formulas) are determined based on the sensitivity levels of the sensors and the noise level in the environment. We determine the threshold as $1°\mathtt{C} = 1.8°\mathtt{F}$ for the temperature sensor, `10 lux` for the illuminance sensor, and `55 dB` for the sound sensors.