Peeves: Physical Event Verification in Smart Homes

Simon Birnbach, Simon Eberz, Ivan Martinovic
Department of Computer Science, University of Oxford
firstname.lastname@cs.ox.ac.uk

ABSTRACT

With the rising availability of smart devices (e.g., smart thermostats, lights, locks, etc.), they are increasingly combined into "smart homes". A key component of smart homes are event sensors that report physical events (such as doors opening or the light turning on) which can be triggered automatically by the system or manually by the user. However, data from these sensors are not always trustworthy. Both faults in the event sensors and involvement of active attackers can lead to reporting of events that did not physically happen (event spoofing). This is particularly critical, as smart homes can trigger event chains (e.g., turning the radiator offwhen a window is opened) without involvement of the user.

The goal of this paper is to verify physical events using data from an ensemble of sensors (such as accelerometers or air pressure sensors) that are commonly found in smart homes. This approach both protects against event sensor faults and sophisticated attackers.

In order to validate our system's performance, we set up a "smart home" in an office environment. We recognize 22 event types using 48 sensors over the course of two weeks. Using data from the physical sensors, we verify the event stream supplied by the event sensors. We consider two threat models: a zero-effort attacker who spoofs events at arbitrary times and an opportunistic attacker who has access to a live stream of sensor data to better time their attack. We achieve perfect classification for 9 out of 22 events and achieve a 0% false alarm rate at a detection rate exceeding 99.9% for 15 events. We also show that even a strong opportunistic attacker is inherently limited to spoofing few select events and that doing so involves lengthy waiting periods.

CCS CONCEPTS

• **Security and privacy** → *Intrusion detection systems*; *Distributed systems security*; *Mobile and wireless security*.

KEYWORDS

Internet of things; smart home; event verification

ACM Reference Format:
Simon Birnbach, Simon Eberz, Ivan Martinovic. 2019. Peeves: Physical Event Verification in Smart Homes. In *2019 ACM SIGSAC Conference on Computer and Communications Security (CCS'19), November 11–15, 2019, London, United Kingdom.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3319535.3354254

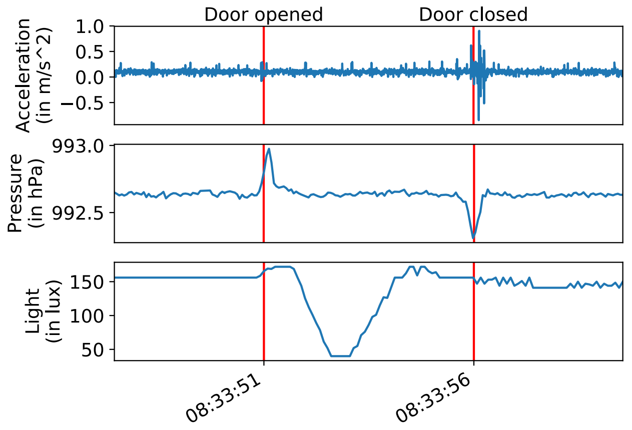


Figure 1: An example of how door events can affect different sensors. The red lines correspond to the timing offi rst a door opening event and then a door closing event. One can see that the door closing event has a clear effect on all three sensors, whereas the door opening event affects primarily the light and pressure sensors.

1 INTRODUCTION

With the advent of commercial smart home devices, the Internet-of-Things (IoT) paradigm is no longer a dream of the future. From smart locks over smart surveillance cameras to remote-controlled garage doors, IoT devices are starting to permeate consumer homes, enriching them with a wide variety of functions and information that is only a button press away.

Unfortunately, this new and fast-evolving market has pushed manufacturers towards quick development cycles that favor features and a fast time to market over security issues. This has led to an ecosystem where vulnerable devices are far-spread and often remain unpatched [18].

Several approaches have been proposed to reduce the risk posed by insecure IoT devices. Most of these approaches focus on either device and smart application permission systems [12, 17, 25] or network traffic analysis [2, 21, 30]. But none of these approaches can protect event integrity after a device has been compromised. A compromised device can control what data it sends to the smart home system and thus lie about what it is really doing or sensing. Possible attacks range from smart cameras being turned on without the knowledge of the occupants to smart locks not locking the door as instructed. An interesting property of smart home systems is the interconnectedness of events. A device that seems to have no security relevance can become an attack vector if it triggers actions by other devices (e.g., a system that unlocks the back door when the kitchen light is being turned on [7]). An overview of smart home devices and potential attacker goals can be found in Table 1.

While the presence of cheap and possibly insecure devices in their homes puts consumers at risk, the pervasiveness of sensors also provides new ways to strengthen the security of these systems. As sensors can measure the physical effects of device actions and events, they can thus be used as a secondary source to verify that these events have actually occurred. As an example, Figure 1 shows how a closing door affects a pressure sensor, a light sensor and an accelerometer placed within the same room. Table 1 provides a (non-exhaustive) overview of which sensors are expected to sense effects of events relating to current smart home devices.

In this work, we introduce Peeves, a system that automatically verifies events in smart homes based on their physical signatures. As events are fed into the system, it uses the distributed sensor data for supervised learning of the corresponding event signatures. By leveraging sensors available in the same physical environment, Peeves can verify if reported events have actually occurred. Our system is able to automatically learn these signatures and the sensors that are relevant for verification without supervision by the user. We show the feasibility of Peeves in a real-world experiment.

Contributions.

- We introduce Peeves, a system to automatically learn smart home event signatures for various sensing modalities.
- We evaluate Peeves in a real-world experiment with 48 sensors and 22 different event types over two weeks. We make this dataset publicly available [1].
- We introduce a method for the attacker to predict spoofing opportunities to increase their success rate and show that Peeves is still effective against this strong adversary.

The remainder of the paper is organized as follows: Section 2 gives an overview of smart home background and related work. Sections 3 and 4 outline our system and threat model. We describe our experimental design in Section 5 and our methods in Section 6. We present our results in Section 7 and conclude the paper in Section 8.

2 BACKGROUND

2.1 Smart home systems

Smart home systems enable users to remotely control and monitor devices and automate everyday tasks. These devices range from surveillance cameras over smart locks to self-learning thermostats. Due to these devices, users are now able to remotely view their homes, automatically unlock the front door when arriving at home, and offload the burden of worrying about energy-efficient heating.

There are many competing smart home systems ranging from commercial solutions such as Samsung's SmartThings [16] and Apple's HomeKit [15] to open-source tools like OpenHAB [24] and HomeAssistant [3].

Even though these systems use different terminology, they all have the same fundamental conceptual design. Each system uses a smart hub as a central node that aggregates data and controls the interactions between devices. Additionally, the hub often connects devices to external cloud services. Devices can have sensors measuring their physical environment, as well as actuators that interact with it. For example, the opening and closing of a door can be detected by a magnetic contact switch (*sensor*) mounted on the doorframe, whereas a smart light switch (*actuator*) can be used to remotely control a ceiling light.

These devices have internal states that are representations of the actual physical states they are monitoring. In the previous example, the door can be either "open" or "closed", while the ceiling light has the states "on" or "off". *Physical events* are state changes that can be triggered by the user directly (e.g., opening a door), through a user interface (e.g., "Alexa, turn on the kitchen light"), or automatically by the system hub (e.g., lower blinds at sunset every day). When a smart device notices that a monitored physical state changes, the device notifies the hub by sending an *event notification*. Note the difference between physical events and event notifications. Physical events are the phenomena happening in the real world, whereas event notifications represent the view that a smart device has (or claims to have) of those phenomena.

Automations of devices are configured by the user through rules that typically follow the trigger-action principle. For example, a rule can state that whenever the kitchen door opens (*trigger*), the ceiling light in the kitchen should be turned on (*action*). In many systems, several actions can be grouped together by creating *modes* to simplify rules. If the user wants to open the blinds and make a cup of coffee every time their alarm goes off, they can create a "good-morning" mode that triggers all these actions at a certain time rather than program them all individually.

2.2 Related work

Security of smart home systems. There has been extensive research focusing on the security of smart home systems and devices. Fernandes et al. [11] investigated the popular SmartThings system and uncovered several vulnerabilities due to overprivileged smart home applications. One of these vulnerabilities allows malicious applications running on the smart hub to spoof events as if they were originating from other devices in the network, thus triggering unintended actions such as erroneously setting off afi re alarm. This risk extends even beyond malicious control software as presented in their paper, as compromised devices can easily send false event updates, or suppress the notification of an actual event.

As demonstrated by Sivaraman et al. [31], malicious mobile phone applications can compromise home networks and thus open up smart home systems to attacks from remote attackers over the internet. Ronen et al. [27] showed how physical proximity can be used to compromise smart devices. They exploited communication protocolfl aws present in Philips Hue smart lamps to create a worm that can spread across a city from household to household. In [26], Ronen and Shamir further showed how attackers can misuse compromised smart devices such as smart lights and go beyond their intended functionality.

Security of trigger-action programming. Previous work has highlighted the risks posed by unsafe and insecure interactions between devices and smart applications. Device actuations can lead to unintentional consequences if trigger-action rules are being used in different contexts [17], if they include external trigger-action platforms [7], or if they include hidden inter-app interactions

[1]https://www.doi.org/10.5287/bodleian:mv22Jk2Xj

Table 1: Smart home devices and possible attacker goals. Attacker goals follow the classification by Denning et. al. [9]. Each device affects the readings of nearby sensors that could be used for event verification. Devices used in our real-world experiment are highlighted in bold.

Device	Attacker goal	Sensors for verification	References
Door	Enabling physical entry, Physical theft	Accelerometer, Light, Air pressure, Microphone	[7]
Window	Enabling physical entry, Physical theft	Accelerometer, Air quality, Air pressure, Microphone	[10, 17]
Window shade	Espionage, Voyeurism	Accelerometer, Power, Microphone, Light	[5, 22]
Light	Data exfiltration, Espionage, Causing physical harm	Light, Power	[26, 27]
Smart cam	Espionage, Voyeurism	Power, Accelerometer	[8]
Thermostat	Enabling physical entry, Extortion	Temperature	[9, 17]
Door lock	Enabling physical entry, Physical theft	Accelerometer, Microphone	[14]
Siren	Causing physical harm, Causing environment damage, Misinformation	Light, Microphone	[11]
Kettle, Oven, Iron	Causing physical harm, Causing environment damage	Power, Temperature, Humidity, Accelerometer, Microphone	[7, 10]

through physical channels [10]. These issues have been addressed through morefi ne-grained permission systems [17, 32], better logging for forensic purposes [33], and static analysis of smart applications [6, 7, 10, 23] to detect possible physical channel interactions. While these systems help to reduce the risk that these interactions pose, they do not protect against cases where an existing and desired trigger-action rule is being exploited, e.g., by a compromised smart home device.

Device and behaviorfingerprinting. There has been an increased interest in identifying smart home devices andfi ngerprinting of device behavior. Several solutions have been proposed that identify devices based on their network trafficfi ngerprint [4, 21, 30]. Other work has focused on inferring smart home activities and device event signatures from network traffic to highlight the privacy implications of wireless communication in smart homes [1, 2].

Closest to our work is HoMonit [34], which uses network traffic based eventfi ngerprints to detect event spoofing from misbehaving smart applications. HoMonit builds automata that model the behavior of smart applications and then extracts event signatures from wireless smart home traffic. In combination, these methods enable the detection of misbehaving smart applications if the expected behavior deviates from the actual behavior inferred from the network traffic. However, this does not prevent event spoofing when the attack originates from outside the smart hub, e.g., because of a compromised smart home device. In this case, the event network signature will be present although the triggering event has not occurred.

Event detection with heterogeneous sensors. Recent research has exploited the fact that smart home events have physical effects that can be measured through various sensing modalities. Perceptio [13] builds on the idea of context-based pairing [20, 29] which relies on common sensor measurements to provide the entropy needed to derive symmetric keys. But as devices present in a smart home have diverse sensing modalities, pairing protocols must consider this device heterogenenity in order to be usable. The authors of this paper make the observation that even if devices do not share a common sensor type, they can be affected by the same physical events in different ways. For example, when in use, a coffee machine has a distinct soundfi ngerprint that can be picked up by a microphone. If the coffee machine is connected to a power meter, that device can detect the power usage of the coffee machine. Although those two sensors might measure very different patterns and even register the event at different times, the timespan between two subsequent events is the same. Perceptio uses this inter-event timing information to constructfi ngerprints for each event type witnessed by a sensor. Thesefi ngerprints are then used to provide the entropy for its pairing protocol.

In a non-security context this observation has been used for activity recognition systems. A recent paper aiming to provide general-purpose sensing in home environments is SyntheticSensors [19]. In this paper, synthetic sensors are introduced as an abstraction to physical events to improve the usability of such a system. The system is based on a single sensor board with nine sensors covering twelve different sensing modalities. Individual synthetic sensors are then represented by a base-level Support Vector Machine (SVM), and a global multiclass SVM is trained on the output of all the base-level SVMs. Examples for the 38 synthetic sensors considered in this paper are events that also typically appear in smart home systems, such as "Kettle on", "Door closed" or "Dishwasher running". This paper gives an interesting outlook on how and what kind of events can be detected in a smart home setting.

However, unlike Peeves, it is not designed with an adversary in mind and events have to be labeled explicitly by the user. In contrast, Peeves can verify the veracity of reported smart home events to defend against event spoofing attacks and uses smart home event notifications to learn event signatures automatically.

3 SYSTEM MODEL

We consider a system model inspired by contemporary smart home systems as described in Section 2. These systems are controlled by a central hub, and all their traffic is routed through that hub.

Besides the hub, Peeves consists of several heterogeneous smart home devices. Devices have attributes that describe what they can do. These attributes can be possible device states or measurements, and supported actions.

In our system, we differentiate between two different attribute types. Attributes related to physical events are considered to be *event sources*. These attributes consist of a finite list of possible device states or actions. For example, a door sensor has the states "open" and "closed", whereas a window shade understands the actions "up" or "down".

When the state of a device *changes* in the real world, we refer to it as a *physical event*. For example, this can be someone turning the door handle and pushing the door to open it, or the window shade moving from one position to another. We define *event notifications* as reported state changes occurring at time t. An event notification is represented by a tuple $E_t = (d, a)$, where d is the affected device and a is the changed attribute of the event source. We assume that events are reported in a timely manner, i.e., the system rejects delayed event notifications.

The goal of Peeves is to verify if the physical event associated with a received event notification did in fact occur. Hence, the verification of a physical event is triggered by the reception of an event notification.

Other sensors of various datatypes provide measurements and are used as *verification sensors*. Data collected by verification sensors gets transmitted to the central hub for further processing. This data is then being used to learn the sensor-specific physicalfingerprints of events. Event verification is done by the hub and is based on the aggregate data of all the sensors. We assume that event sources remain uncompromised during training and that the hub system software and verification sensors remain uncompromised during operation.

Peeves provides a countermeasure to *event spoofing*, i.e., when an event source or a malicious application running on the hub [11] reports an event that has not physically happened. This should not be confused with *event masking*, i.e., when no event notification is sent for a real physical event. We leave the exploration of event masking for future work. An overview of Peeves is given in Figure 2.

4 THREAT MODEL

We consider an attacker that is able to remotely compromise event sources in the smart home. This allows them to trigger any supported actions or events directly, bypassing the hub. The attacker is, however, not able to make any physical changes to the devices or events. Furthermore, the attacker is able to choose when and if

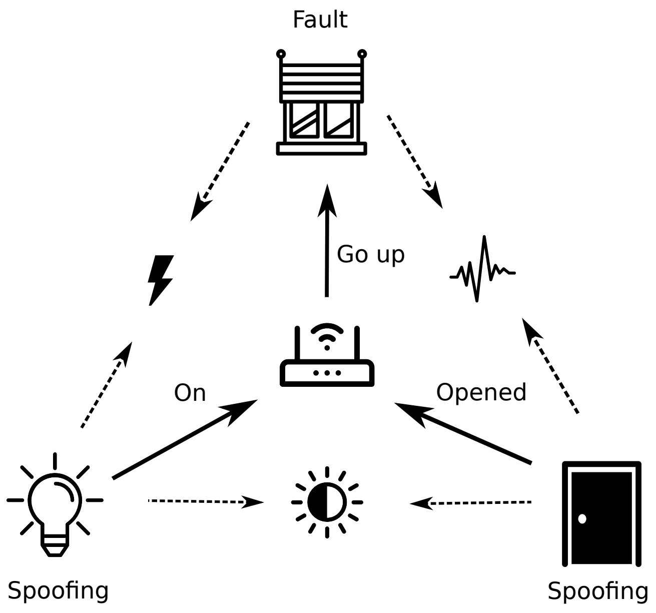


Figure 2: Thisfi gure gives an overview over Peeves and three exemplary scenarios. The window shade at the top exhibits a fault and does not act on a command by the smart hub in the middle. On the bottom, the door sensor and light bulb send a spoofed event to the hub, falsely claiming to have been triggered. Peeves uses verification sensors to check the physicalfi ngerprints of the corresponding events. This is possible as these events have measurable effects (dashed lines) on sensors measuring physical phenomena such as power meters (top left), accelerometers (top right) or light sensors (bottom).

they report an event to the hub. However, events that have a large delay between event timestamp and time of notification will be automatically rejected by the system.

For example, an attacker might want to gain access to a house. Similar to the example in IoTGuard [7], the back door of the house gets unlocked when the smart home enters a "home mode" which is triggered when the front door opens. The attacker can then use a compromised door sensor to *spoof* an opening door event to trigger the "home mode" and thus unlock the back door.

However, mismatches between the internal state of a smart home system and reality are not limited to malicious behavior. In our experiments, we have experienced how device faults can have similar effects to attacks. Examples include a window shade not executing commands despite acknowledging them (e.g., due to being stuck) and a smart radiator valve that did not notice that it was constantly heating due to a blocked actuation pin. These kinds of faults are particularly common with "retrofitted" smart devices, i.e., regular devices with added smart control logic. The less integrated the smart and normal components of the device are, the more likely is a mismatch between the device's reported and actual state changes.

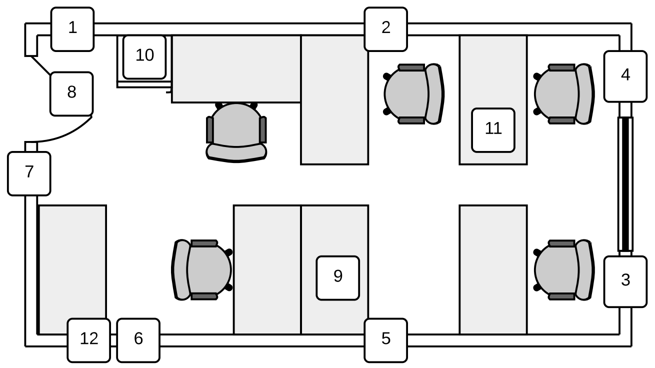


Figure 3: The device deployment for the experiment is pictured in thisfl oorplan. The numbers refer to the group numbers in Tables 3 and 4. Sensor group 8 is mounted on the door, groups 9 and 11 are placed on tables, and group number 4 is fixed to a shelf. All the other sensors are mounted on the wall.

Our system considers two types of attackers:

- *Zero-effort attacker/Sensor fault*
- *Opportunistic attacker*

For both types of attackers, event sensors are considered to be untrusted, as they can be spoofed or exhibit random faults.

The zero-effort attacker spoofs events at arbitrary times and does not have knowledge of any system parameters such as the features used for detection. Furthermore, this attacker can not read the data of the verification sensors.

In comparison, the opportunistic attacker knows all the system parameters such as the features used for detection and has read access to the current data of all sensors. Using this information, this attacker tries to spoof only at times when they think they will be likely to succeed. The attacker's goal is to maximize the probability of their spoofed event going undetected (i.e., being incorrectly judged as genuine) while minimizing the amount of time they have to wait for a spoofing opportunity. Neither attacker has physical access to devices and we assume attackers can not modify or suppress data originating from verification sensors.

5 EXPERIMENTAL DESIGN

To evaluate the feasibility of our proposed system, we conduct an experiment in which we collect the physical signatures of 22 event types in a smart office environment. The experiment is conducted continuously over two weeks while the office is being used by four people for their everyday work.

5.1 Overview

During the study, we collect data for a diverse range of events and sensing modalities. In an office, we distribute thirteen Raspberry Pis, each equipped with several environmental sensors to collect data. The deployment for this experiment can be found in Figure 3, while a detailed sensor configuration is listed in Table 3. A description of the sensor types used in the experiment can be found in Table 2. We consider as event sources: an office door with automatic closing mechanism, a sliding window, a ceiling light, an automatic window shade, a fridge, an automated tower fan, a smart radiator valve, a smart coffee machine, a PC with attached screen and a smart camera with privacy mode. Over the whole two week duration of the study, we registered 2773 events in our smart environment. A description of the event sources together with their corresponding events and event occurrences can be found in Table 4.

Table 2: The different sensor modality types considered as verification sensors in the real-world experiment. The number of sensors deployed of a certain type is given in parentheses.

Abbreviation	Sensor modality (#)
A	Accelerometer (10)
G	Gyroscope (10)
M	Magnetometer (1)
P	Air pressure (9)
H	Humidity (5)
T	Temperature (10)
L	Light (7)
PM	Power meter (7)
AQ	Air quality/CO_2 (2)
S	Sound pressure level (6)
R	Received signal strength (6)
TC	Thermal camera (2)

5.2 Data collection

The ground truth for the events is collected as follows. The door, window and fridge are outfitted with magnetic contact sensors, shown in Figure 4a. Light switch events are registered through a force-sensitive resistor that registers when the switch is being pressed, see Figure 4b. The smart cam follows a schedule, periodically switching between privacy mode and normal mode every hour. When the smart cam is in privacy mode, its camera is tilted towards its case, preventing it from recording video from the room. Window shade, radiator, fan and coffee machine can be controlled via a tablet, see Figure 4c. As the fan used in the experiment does not have any network connectivity, we automate it by using a Raspberry Pi Zero as an infrared remote control, shown in Figure 4b. The doorbell events are being polled from the Ring servers.

Events are triggered sporadically by the office occupants and automatically at certain times. Smart cam, PC and screen events are only triggered automatically, whereas door, window, fridge, doorbell, and coffee machine events are only triggered manually. Automatic events follow a schedule with randomized starting times to avoid systematic cross-contamination of event signatures. More details on event schedules are given in Table 5.

We use the Network Time Protocol (NTP) to synchronize time across the Raspberry Pis used in the experiment. The sensors connected to the Raspberry Pis are continuously being polled over their I^2C interface. The measured data is being saved locally on USB sticks. As an example, the sensor configuration of one Raspberry Pi is shown in Figure 4a.

Table 3: The sensor configuration for the real-world experiment. The sensor modalities covered by each sensor are given in parentheses. The used abbreviations can be found in Table 2. The deployment is pictured in Figure 3.

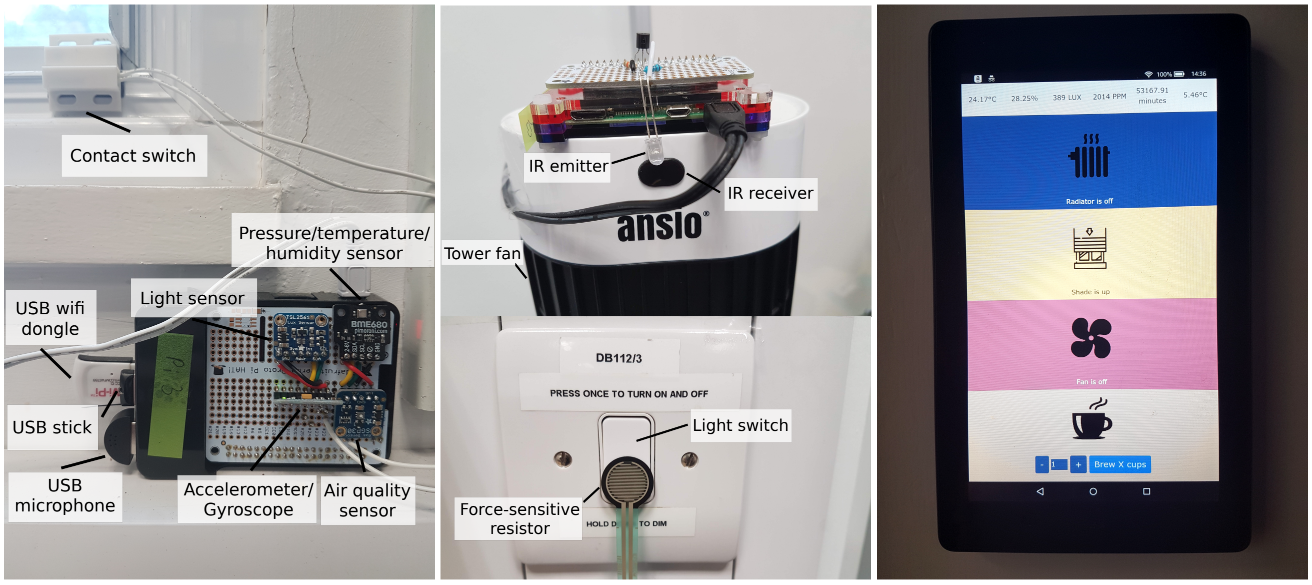
Group #	Device	Sensor (Modalities)
1	Pi #1	MPU6050 (A/G) BMP280 (P/T) TSL2560 (L) USB microphone (S) Ralink RT5370 USB WiFi dongle (R)
2	Pi #2	MPU6050 (A/G) BMP280 (P/T) TSL2560 (L) USB microphone (S) Ralink RT5370 USB WiFi dongle (R)
3	Pi #3	MPU6050 (A/G) BME680 (P/H/T) TSL2560 (L) SGP30 (AQ) USB microphone (S) Ralink RT5370 USB WiFi dongle (R) Contact sensor (window 2)
4	Pi #4	MPU6050 (A/G) Contact sensor (window 1)
	Plug #3	TP-Link HS110 (PM of window shade)
	Plug #4	TP-Link HS110 (PM of smart cam)
5	Pi #5	MPU6050 (A/G) BMP280 (P/T) TSL2560 (L) USB microphone (S) Ralink RT5370 USB WiFi dongle (R)
6	Pi #6	MPU6050 (A/G) BMP280 (P/T) TSL2560 (L) USB microphone (S) Ralink RT5370 USB WiFi dongle (R)
7	Pi #7	MPU6050 (A/G) BME680 (P/T/H) TSL2560 (L) SGP30 (AQ) USB microphone (S) Ralink RT5370 USB WiFi dongle (R) Force-sensitive resistor (ligh switch)
8	Pi #8	ST Micro LPS25H (P/T) ST Micro HTS221 (H/T) ST Micro LSM9DS1 (A/G/M) Contact sensor (door)
9	Pi #9	MPU6050 (A/G)
	Pi #13	Fan remote control
	Plug #5	TP-Link HS110 (PM of fan)
	Plug #6	TP-Link HS110 (PM of PC)
	Plug #7	TP-Link HS110 (PM of screen)
10	Pi #10	MPU6050 (A/G on fridge) BME680 (A/T/H on fridge) BME680 (A/T/H inside fridge) TSL2560 (L inside fridge) Contact sensor (fridge)
	Plug #1	TP-Link HS110 (PM of fridge)
	Plug #2	TP-Link HS110 (PM of coffee machine)
11	Pi #11	MLX90640 (TC)
12	Pi #12	MLX90640 (TC)

Table 4: This table shows the event sources considered in the real-world experiment and their corresponding event notifications. The number of event occurrences is given in parentheses. The group number refers to the deployment group number pictured in Figure 3.

Group #	Event source	Event notifications (#)
3	Window	Window opened (44) Window closed (44)
	equiva eQ3 radiator thermostat	Radiator on (41) Radiator off(41)
4	Samsung SmartCam SNH-VP6410PN	Camera on (156) Camera off(156)
	Teptron Move window shade controller	Shade up (74) Shade down (75)
7	Light switch	Light on (47) Light off(48)
8	Door	Door opened (340) Door closed (340)
	Ring doorbell	Doorbell used (52)
9	Desktop PC	PC on (73) PC off(72)
	LCD screen	Screen on (144) Screen off(213)
	Tower fan	Fan on (334) Fan off(335)
10	Mini fridge	Fridge opened (56) Fridge closed (56)
	Smarter Coffee	Coffee machine used (34)

Table 5: This table gives an overview over which events were triggered manually or automatically. For automatically triggered events, the schedule of device actuations is given.

Event source	Manual	Automatic	Schedule
Camera	✗	✓	Toggle every hour (24/7)
Coffee machine	✓	✗	–
Door	✓	✗	–
Doorbell	✓	✗	–
Fan	✓	✓	Night: 2 min every 30 min day: 2 min every 2 h
Fridge	✓	✗	–
Light switch	✓	✗	–
PC	✗	✓	2 h on, 2 h off(24/7)
Radiator	✓	✓	Twice per day for 30 min once for 15 min
Screen	✗	✓	30 min on, 30 min off during PC duty cycle
Window shade	✓	✓	6-9am: toggle every 30min 7-9pm: toggle every 30min
Window	✓	✗	–



(a) The sensor configuration for one of the Raspberry Pis. A detailed listing of the sensor configuration can be found in Table 3. The contact switch at the top is being used to detect when the window is being opened or closed.

(b) The fan can be remote controlled through a Rasberry Pi Zero acting as an IR remote. Light switch presses are recorded through an attached force-sensitive resistor.

(c) The user interface running on an Amazon Fire tablet. It allows users to control the fan, radiator and window shade, as well as make coffee.

Figure 4: Experimental setup

In addition to the sensors connected to the Raspberry Pis, we use TP-Link HS110 power meters to measure the voltage, current and power usage of the corded devices. The measurement data of the power meters is continuously polled over WiFi.

5.3 Ethical concerns

This project has been reviewed by and received clearance by the responsible research ethics committee at our university. The main concerns relate to the door contact sensors, thermal cameras and the microphones used as sound pressure sensors. The door contact sensors could, in theory, be used to track people's movement (i.e., when they enter or leave the office). However, as the office is used by multiple people, it would not be possible to relate this data to individuals. In addition, individuals can enter or leave the room freely if the door is propped open. Naturally, microphone data would be far more sensitive. To avoid this issue, we use the microphones only as sound pressure sensors that record sound levels, rather than the raw audio. As such, it is impossible for sensitive audio data (e.g., conversations) to be recorded. Raw sound pressure values allow the detection of noise, but they do not allow one to accurately determine the event or activity that caused it. Despite their low resolution, the thermal cameras could be used as low-quality video cameras, as the outlines of humans can be easily distinguished from the colder environment. Therefore, we only collect column and row averages of thermal image frames and thus make it impossible to recreate the original video data.

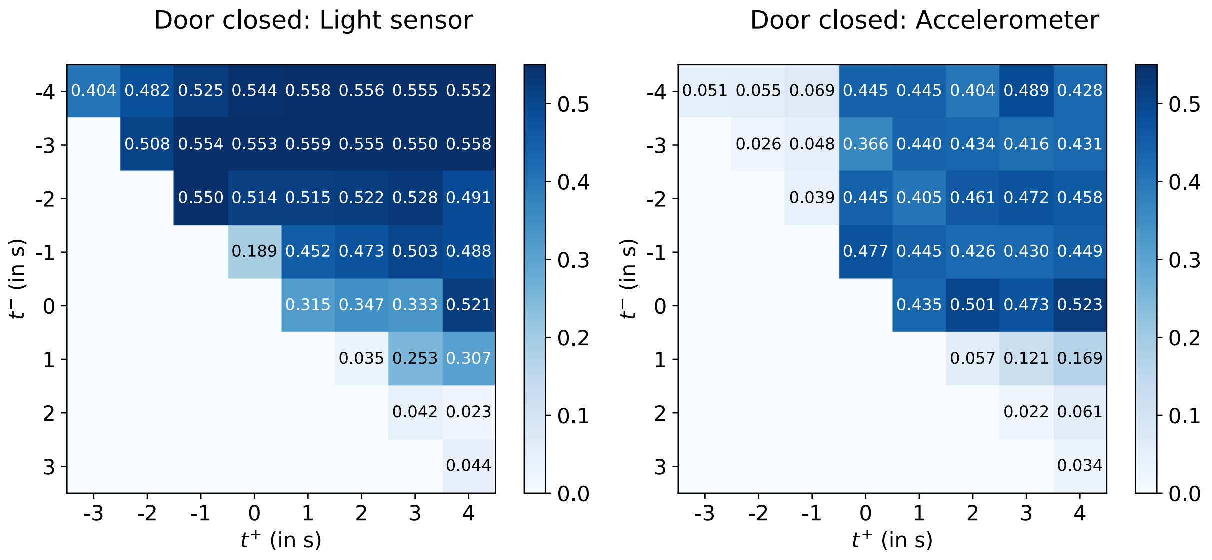
6 METHODS

6.1 Data labelling

Separately for each event, we label all points in time with either 0 (no event occurred) or 1 (event occurred). The timestamps of 1-samples are identified by the corresponding ground truth (e.g., the precise moment a door closes as identified by the contact switch). The event time for automatically triggered events is the moment the command is sent to the device. 0-events are generated periodically in 1-second intervals for the remainder of time. This approach initially leads to "ambiguous" labels close to events. For example, if a door closes at time t, the system will likely detect (and correctly verify) this in a small window around t. The fact that a 0-sample very close to a 1-sample is classified as 1 (i.e., potential spoofing opportunity) does arguably not pose a problem. While this would, in theory, allow the attacker to spoof events just before or after actual events, there would be no incentive to do so. In order to deal with this, we use so-called safety margins around events. We remove samples measured during such a safety margin from both the training and test data. We set the size of the safety margin to twice the size of the largest sensor window.

6.2 Dataset division

We split our 2-week dataset into three parts: development, training and testing. Each of these parts forms a block of consecutive samples, rather than choosing the appropriate fraction of samples randomly over the entire dataset. This reflects actual system operation, for which the training phase has to predate testing in its



(a) Light sensor

(b) Accelerometer

Figure 5: Window parameter grid search for the "Door closed" event. Values given are the average RMI.

entirety. This is especially crucial as we require the event sources to remain uncompromised during the training phase. We use thefirst day of data for development and split the remaining data evenly into training and testing sets.

The development set is used for two purposes: Calculation of the sensor-specific windows and feature selection. We describe the methods for both of these in the following subsections. Based on the selected features, we then train and test the classifier on the remaining two parts.

6.3 Sensor window selection

Each event is associated with a single timestamp (e.g., the precise moment the contact sensors detects a closing or opening door). However, the physical signature of the event can precede, follow or overlap with the event. An illustration of these differences is shown in Figure 1. Even for the same event, the time window in which the event signature is noticeable depends on the sensor. For example, the door closing is preceded by changes in light and followed by an accelerometer response. This event signature window defines the period over which sensor data is collected to verify the event (see the next subsection for details on feature extraction). While it would be possible in principle to manually set these sensor windows for each event based on the data, we develop an automatic approach that is used before the system's training phase:

The window size and position relative to the event is controlled by two parameters: t^+ and t^-. Given an event timestamp t, the corresponding window will be the interval $(t - t^-, t + t^+)$. For each event, we obtain the optimal values for both parameters through a grid search on the development set. The performance indicator for this grid search is the average relative mutual information (RMI, see next subsection) of all features. In order to avoid distortion of the average by very poor features, we exclude features that do not perform better than a "random" feature. An illustration of the results of this grid search for the same event and two different sensors can be seen in Figure 5. It is evident that the window for the light sensor largely precedes the event (i.e., negative values of t^-) whereas the accelerometer response achieves the best distinctiveness in a 4-second window directly following the event.

6.4 Feature extraction

The baseline values of many of our environmental sensorsfluctuate heavily throughout the day (e.g., the air pressure readings are dependent on the ambient air pressure outside). As we still want to capture transient changes caused by events, we have to compute changes relative to the current baseline. To do so, we compute the features in the following manner. For each 0-sample and each 1-sample (see Section 6.1), we construct the event signature window as described in the previous section using parameters t^+ and t^-. As the current baseline, we subtract the mean of the preceding window with the same length as the event signature window. We then computefi ve statistical features (min, max, mean, sum, and standard deviation) on the corresponding event signature window on the data of every sensor.

Following the feature extraction, we measure each feature's quality (i.e., distinctiveness). This is needed for the computation of ideal window sizes for each sensor (see following subsection) and feature selection. We use relative mutual information (RMI) as a quality measure. RMI is defined as

$$\mathrm{RMI}(event, F) = \frac{H(event) - H(event|F)}{H(event)}$$

where H(A) is the entropy of A and H(A|B) denotes the entropy of A conditioned on B. *event* denotes the ground truth for this sample (i.e., 0 or 1) and is therefore discrete. However, the feature space for most features is continuous. In order to precisely determine the conditional entropy, binning would be required to discretise the features. However, the reported RMI would depend on the binning strategy and number of bins (with more bins leading to a higher calculated RMI). To avoid this problem, we use the non-parametric approach proposed by Ross et al. to estimate the mutual information between the discrete event ground truth and continuous features [28].

Due to the small number of 1-samples and the resulting class imbalance, features often show a comparatively high RMI in the development set without actually providing systematic information about the event. This is particularly critical as feature selection is performed on a small subsample of the entire data. Due to timed automated events, events sometimes occur at the same time. While this is not systematic, even a small number of overlaps can lead to overstated RMI. In order to limit the effect of noisy features and reduce model training time, we select features with an RMI above 40%.

6.5 Classification and metrics

We use a binary linear support vector machine (SVM) as our classifier. We choose a linear kernel due to the large amount of training data (roughly 600,000 samples) and high number of features. The classifier is parametrised by the penalty parameter C. Since the two classes (events and non-events) are highly imbalanced, we assign different values of C proportional to the fraction of samples for each class. This avoids biasing the classifier towards the more common class (i.e., non-events).

Instead of using the actual binary classifier decision, we calculate each sample's distance to the decision boundary. This allows us to then define a threshold on this distance to control the tradeoff between detection rate (DR) and false alarm rate (FAR). The DR is defined as the fraction of 0-samples that are correctly classified (i.e., the probability of detecting a spoofing attack conducted at a random time). Conversely, the false alarm rate is defined as the fraction of wrongly classified 1-samples (i.e., actual events that are wrongly classified as spoofed). We combine both metrics into the equal error rate (EER), which is the error rate at a threshold where ($FAR = 1 - DR$). Since the relatively limited number of 1-events leads to equally limited number of possible FARs, we use geometric interpolation (see Figure 7) to obtain the EER.

6.6 Opportunistic attacker

As outlined in our threat model (Section 4), we consider two kinds of attackers: zero-effort attacker and opportunistic attackers. The zero-effort attacker chooses their moment for event spoofing randomly without consideration for the physical environment. Conversely, the opportunistic attacker uses information about the verification sensors to spoof at times when they are particularly likely to be successful.

The goal of the opportunistic attacker is to predict whether a given point in time will be misclassified by the verification system (i.e., at what time a spoofed event will be incorrectly classified as genuine). Intuitively, the attacker could copy the verification system since they have access to sensor readings and ground truth. Based on this system, they will learn the system's score for each potential point in time. However, this will only grant them posterior knowledge about opportunities (i.e., they will learn that a good spoofing opportunity would have been one second ago). In order to judge whether a time t is suitable for spoofing, the attacker can only use information collected up to time t. To reflect this limitation, we build an attacker *surrogate model* as follows:

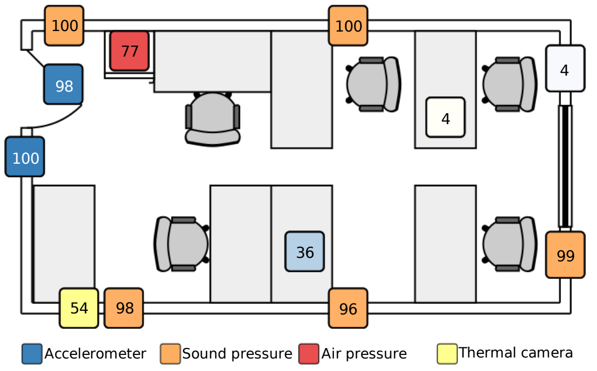
Table 6: Maximum RMI for each event within the different sensor groups. Sensor group labels are given in Table 2.

Event	A	G	M	P	H	T	L	PM	AQ	S	R	TC
Coffee machine used	87	67	4	12	11	24	82	100	75	37	9	37
Doorbell used	43	50	45	30	25	33	65	37	70	38	74	24
Door closed	99	96	96	88	41	69	99	35	41	100	66	53
Door opened	85	93	96	75	36	61	86	41	57	80	55	17
Fan off	24	11	2	2	2	53	9	100	12	62	27	11
Fan on	25	5	2	2	2	21	7	100	42	47	41	10
Fridge closed	87	77	5	98	54	92	99	45	33	73	61	67
Fridge opened	64	67	3	21	47	96	99	49	38	66	57	67
Light off	55	47	40	36	7	25	100	55	26	79	65	28
Light on	51	56	51	41	6	31	100	57	24	32	62	47
PC off	10	7	3	6	3	21	6	99	24	6	77	9
PC on	15	11	5	6	5	30	7	100	49	8	6	6
Camera off	7	5	3	4	4	7	18	99	17	5	40	5
Camera on	4	4	6	3	2	9	16	99	18	5	40	7
Radiator off	10	24	5	14	5	41	25	57	10	21	19	16
Radiator on	12	8	8	9	7	32	15	76	4	12	14	79
Screen off	6	3	4	5	3	14	22	99	15	4	38	4
Screen on	6	4	2	3	6	15	17	100	18	8	44	7
Window shade down	90	79	5	11	7	23	12	93	51	24	12	40
Window shade up	81	65	6	4	10	22	17	100	53	23	55	51
Window closed	16	20	11	19	5	12	91	64	35	75	95	34
Window opened	9	32	11	45	4	11	96	65	41	99	83	36

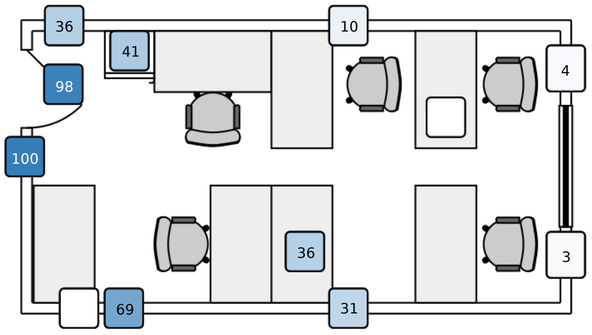
The attackerfi rst uses the development dataset to calculate the ideal windows for each sensor. In line with the attacker's limitations, they restrict the t^+ parameter to non-positive values. Intuitively, this means that only sensor information collected before the "event" is used. They then follow the same methods as the actual system (i.e., feature selection and classifier training). Ideally, the scores produced by the surrogate model will be correlated with those of the actual system. The attacker can then continuously classify samples when no events are happening tofi nd an opportunity (i.e., a sample that scores well on their model). The attacker can then define a threshold for a spoofing attack. This threshold controls the tradeoff between success rate (i.e., the probability the system will judge the attacker's spoofing as a genuine event) and the waiting time until an opportunity is found.

7 RESULTS

Feature distinctiveness. Table 6 shows the distinctiveness of feature groups for each event. Each cell shows the maximum RMI within the sensor group. Most results are relatively intuitive: Accelerometers detect door events, window shade events and the use of the coffee machine. Power meters primarily detect the devices they are connected to. Light sensors detect both toggling of the light switch and events that lead to obscuring the sensor (e.g., a door opening in the proximity of the sensor). Signal strength (RSS)



(a) Most distinctive sensor based on their position. The number given for each sensor is their maximum RMI in %.



(b) Relative importance of accelerometer sensors depending on their position. Numbers are maximum RMI in %.

Figure 6: Type and quality of most distinctive sensor depending on its position

changes are particularly useful to detect window events. This is a consequence of windows only being operated manually. To do so, the user obscures the line of sight (LOS) between the sensor and the router, which causes predictable RSS changes. RSS changes are not only caused by LOS breaks, but also interference which makes RSS useful to detect the PC's power state changes. Note, that RMI is used to determine the distinctiveness of each individual sensor. Hence, the system is not dependent on a specific sensor configuration or deployment, but is instead able to learn relevant sensors in situ.

One of the more surprising results is the high distinctiveness of power meter features for virtually all events. Naturally, this effect is expected for devices that are attached to individual power meters (i.e., fan, fridge, PC, screen, window shade, camera). With the exception of the fridge, the power meters pick up the spike in power once the device is turned on. When the fridge is opened and the temperature rises, the increased cooling leads to a (delayed) increase in power consumption. We observed a voltage spike in all power meters that occurs when the light is turned on or offwhich explains the relatively high RMI for these two events. This effect is particularly pronounced when the overall electrical activity is low. These results show that power meters are useful even beyond detecting state changes of their respective devices.

Figure 6a shows how different sensors contribute to detecting the door closed event. Sound pressure sensors achieve very high RMI that is independent from the sensor's position. Interestingly, the air pressure sensor *inside* the fridge achieves a similarly high RMI. Our inspection of the raw signal suggests that the door closing actually causes a pressure change inside the fridge and that the reading is not a technical artifact (such as vibration).

Figure 6b shows the importance of accelerometers in detecting the door closed event depending on the position of the individual sensor. This importance is not purely down to distance from the door. In fact, sensors on the lock side of the door show higher RMI despite being further away. Unlike sound pressure, the accelerometer signal goes virtually undetected at the other end of the room.

Table 7: Error rates for the detection of different events. Numbers in brackets signify the absolute number of undetected events.

Event	#Events	EER	DR(FAR=0)	FAR(DR=99.9%)
Coffee Machine used	20	0.00	100.00	0.00 (0)
Door closed	177	0.00	100.00	0.00 (0)
Door opened	179	0.00	100.00	0.00 (0)
Fan on	165	0.00	100.00	0.00 (0)
Fridge closed	29	0.00	100.00	0.00 (0)
Fridge opened	28	0.00	100.00	0.00 (0)
Light off	26	0.00	100.00	0.00 (0)
PC on	33	0.00	100.00	0.00 (0)
Screen on	70	0.00	100.00	0.00 (0)
Camera on	70	0.01	99.99	0.00 (0)
Light on	25	0.02	99.98	0.00 (0)
Camera off	69	0.10	99.90	0.00 (0)
Fan off	162	0.17	99.82	0.00 (0)
Screen off	105	0.29	99.71	0.00 (0)
PC off	35	0.39	99.61	0.00 (0)
Radiator on	19	0.97	98.93	5.26 (1)
Window opened	24	4.01	0.35	4.17 (1)
Window shade up	33	11.39	0.28	15.15 (5)
Window shade down	32	12.04	0.19	12.50 (4)
Doorbell used	27	15.06	0.36	55.56 (15)
Window closed	24	15.24	17.63	33.33 (8)
Radiator off	21	49.41	7.65	100.00 (21)

Classification results. The classification results are shown in Table 7. For each event, we list its EER, the detection rate at a threshold that does not cause false alarms and the false alarm rate at 99.9% detection rate. For 9 out of 22 events, we achieve perfect classification (i.e., an EER of 0% or a 100% detection rate at 0% FAR). Some of these events (e.g., door events) draw this good performance from being detected by a multitude of sensors. Others (such as light events) have few, but extremely reliable features. In order to more closely investigate events with less ideal performance (such as window events), we show their ROC curve in Figure 7. The ROC shows the

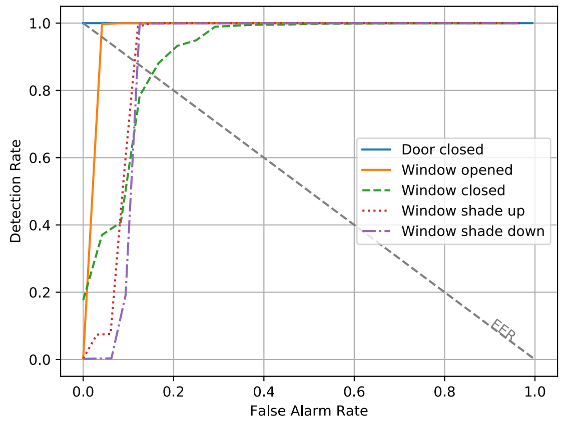


Figure 7: Receiver Operating Characteristics (ROC) curve for different events. Here we only show events with an EER significantly higher than 0 (well-classified events with an EER of 0 are afl at line at the top).

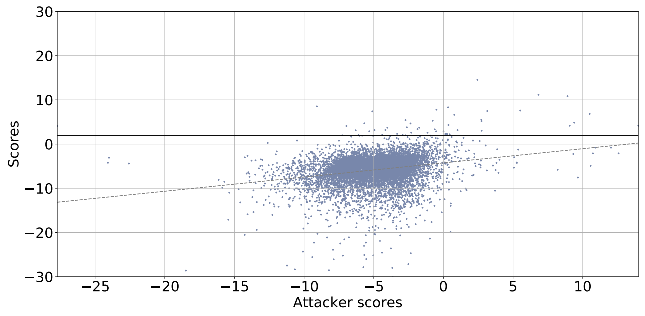


Figure 8: Score correlation between the legitimate model and the opportunistic attacker's surrogate model. The dashed grey line is the linear regression line and the solid black line shows the 99% detection rate cut- off.

tradeoff between DR and FAR. For most events shown in the curve it is evident that few instances of the event were assigned a very poor classification score. If the threshold is adjusted to (incorrectly) reject them as spoofs, the detection rate almost instantly jumps to 100%. This is most likely due to the event being triggered in a way that is very different from the training data (e.g., a person opening a window only an inch). This is a natural consequence of our largely uncontrolled experimental design.

Opportunistic attacker. The goal of the attacker's surrogate model (see Section 6.6) is to accurately predict moments when a spoofed event will be incorrectly verified by the system (we call these moments *opportunities*). Wefi rst measure the quality of the surrogate model by measuring the correlation between the scores produced by the surrogate model and those of the legitimate model. If this correlation is high, the attacker is more likely to correctly identify opportunities. A visualization of this correlation for the window

Table 8: Correlation between the scores produced by the legitimate model and the attacker's surrogate model.

Event	r-value	p-value
Door closed	0.29	0.00
Fridge closed	0.26	0.00
Window closed	0.21	0.00
Screen off	0.18	0.00
Window opened	0.11	0.00
Doorbell used	0.11	0.00
Door opened	0.10	0.00
Screen on	0.02	0.00
Radiator off	0.01	0.00
Radiator on	0.01	0.00
Window shade up	0.01	0.00
Coffee Machine used	0.00	0.04
Window shade down	0.00	0.09
PC off	0.00	0.25
Fridge opened	0.00	0.78
PC on	0.00	0.98
Camera off	-0.00	0.14
Fan on	-0.00	0.88
Camera on	-0.01	0.00
Fan off	-0.04	0.00
Light off	-0.04	0.00
Light on	-0.11	0.00

open event is shown in Figure 8. It is evident that most opportunities (i.e., samples above the 99% DR cut-off) score relatively highly on the attacker's model. A complete list of correlation coefficients (r-values) is given in Table 8. Door, fridge and window and screen events show a particularly high correlation. For these events, we show in Figure 9 how successful the attacker can be depending on how conservatively they choose his opportunity threshold. For the lowest threshold (corresponding to the highest-scoring 0.5% of samples), the success rate increases from the baseline 1% to over 50% for the fridge closed event. While 0.5% would suggest an opportunity roughly every 200 samples, this is not true in practice because opportunities do not arise randomly, but typically occur in bursts. Table 9 shows an analysis of the median time an attacker has to wait for an opportunity depending on the chosen threshold value. For comparison, the zero-effort attacker achieves an average 1% success rate with no need to wait. Note that both Table 9 and Figure 9 are computed for afi xed 99% DR to facilitate comparing events with different EERs.

This analysis shows that opportunities for most events are extremely difficult to predict in a timely manner since their physical event signatures almost exclusively follow the event itself. As a result, the attacker could only detect the opportunity after it happened, at which point it will be too late to spoof an event. In addition, the attacker can only be successful if there are any opportunities *at all*. This means that for events with a 100% detection rate, the attacker will always be unsuccessful.

Table 9: Performance of the opportunistic attacker: For each threshold, the table shows the median time until an opportunity is found (tto) and the corresponding success rate (SR).

	th=0.5%		th=1%		th=5%	
Event	tto	SR	tto	SR	tto	SR
Fridge closed	470	0.56	313	0.38	130	0.10
Screen off	727	0.34	344	0.19	48	0.05
Window closed	1100	0.31	552	0.24	85	0.10
Door closed	1725	0.27	230	0.23	26	0.09
Fridge opened	2443	0.17	965	0.11	129	0.03
Light off	1639	0.16	998	0.10	111	0.04
Doorbell used	6567	0.09	5823	0.07	2915	0.04
Fan off	912	0.08	364	0.11	49	0.08
Light on	540	0.08	383	0.06	151	0.02
Window opened	413	0.04	178	0.03	43	0.02
Window shade up	440	0.03	193	0.02	29	0.01
Door opened	1584	0.02	980	0.02	91	0.02
Fan on	398	0.02	155	0.02	51	0.01
Coffee Machine used	1694	0.01	1264	0.02	80	0.02
PC off	267	0.01	135	0.01	30	0.01
PC on	258	0.01	128	0.01	26	0.01
Camera off	230	0.01	114	0.01	25	0.01
Camera on	245	0.01	126	0.01	24	0.01
Radiator on	1091	0.01	331	0.01	58	0.01
Screen on	400	0.01	175	0.01	33	0.01
Window shade down	227	0.01	115	0.01	26	0.01
Radiator off	595	0.00	224	0.01	44	0.01

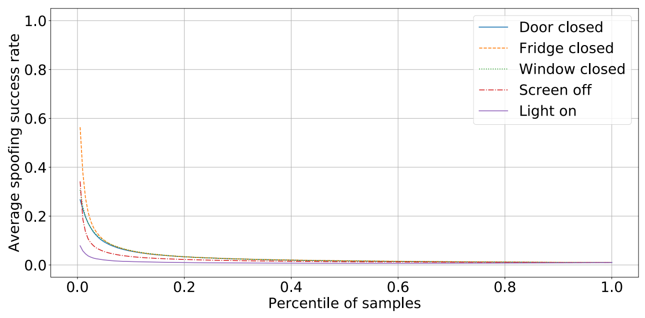


Figure 9: Success rates of the opportunistic attacker for various thresholds. A higher thresholds leads to higher attack success rate but increased wait time for an opportunity.

8 CONCLUSION

In this paper we have presented PEEVES, a system to automatically verify smart home events based on their physical event signatures. We verify 22 events using 48 verification sensors and validate our system on a real-world dataset collected over two weeks.

We show that most events are detected by a multitude of sensors, which highlights that PEEVES can be used even with few off-the-shelf sensors already integrated in smart home devices. 9 out of 22 events achieve a near-perfect EER of 0.00% and 15 events achieve a 0% false alarm rate at a detection rate exceeding 99.9%.

We also formulate the notion of an opportunistic attacker that attempts to predict opportunities (i.e., times when a spoofed event will go undetected) based on verification sensor data. This attacker builds a surrogate event verification model based only on data preceding each sample. We show that some events exhibit signatures that precede or overlap with the event, which makes it possible for the attacker to predict opportunities *before* they occur. We note that spoofing opportunities for the fridge, window and door events are easiest tofi nd, although their high detection rate still makes an attack very difficult to execute.

We make our entire dataset available online to allow researchers to build on our results.

ACKNOWLEDGMENTS

This work was supported by a grant from Mastercard and the Engineering and Physical Sciences Research Council.

REFERENCES

[1] Abbas Acar, Hossein Fereidooni, Tigist Abera, Amit Kumar Sikder, Markus Miettinen, Hidayet Aksu, Mauro Conti, Ahmad-Reza Sadeghi, and A. Selcuk Uluagac. 2018. Peek-a-Boo: I see your smart home activities, even encrypted! *CoRR* abs/1808.02741 (2018). arXiv:1808.02741 http://arxiv.org/abs/1808.02741
[2] Noah Apthorpe, Dillon Reisman, Srikanth Sundaresan, Arvind Narayanan, and Nick Feamster. 2017. Spying on the Smart Home: Privacy Attacks and Defenses on Encrypted IoT Traffic. *CoRR* abs/1708.05044 (2017). arXiv:1708.05044 http://arxiv.org/abs/1708.05044
[3] Home Assistant. 2019. Awaken your home. https://www.home-assistant.io/ [Online; accessed 12-May-2019].
[4] Bruhadeshwar Bezawada, Maalvika Bachani, Jordan Peterson, Hossein Shirazi, Indrakshi Ray, and Indrajit Ray. 2018. Behavioral Fingerprinting of IoT Devices. In *Proceedings of the 2018 Workshop on Attacks and Solutions in Hardware Security (ASHES)*. ACM, 41–50.
[5] Simon Birnbach, Richard Baker, and Ivan Martinovic. 2017. Wi-Fly?: Detecting Privacy Invasion Attacks by Consumer Drones. In *Proceedings of the 24th Annual Network and Distributed System Security Symposium (NDSS)*.
[6] Z Berkay Celik, Patrick McDaniel, and Gang Tan. 2018. Soteria: Automated IoT safety and security analysis. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. 147–158.
[7] Z Berkay Celik, Gang Tan, and Patrick McDaniel. 2019. IoTGuard: Dynamic Enforcement of Security and Safety Policy in Commodity IoT. In *Proceedings of the 26th Annual Network and Distributed System Security Symposium (NDSS)*.
[8] Kaspersky Lab ICS Cert. 2018. Somebody's watching! When cameras are more than just 'smart'. https://ics-cert.kaspersky.com/reports/2018/03/12/somebodys-watching-when-cameras-are-more-than-just-smart/ [Online; accessed 13-May-2019].
[9] Tamara Denning, Tadayoshi Kohno, and Henry M Levy. 2013. Computer security and the modern home. *Commun. ACM* 56, 1 (2013), 94–103.
[10] Wenbo Ding and Hongxin Hu. 2018. On the Safety of IoT Device Physical Interaction Control. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 832–846.
[11] Earlence Fernandes, Jaeyeon Jung, and Atul Prakash. 2016. Security analysis of emerging smart home applications. In *2016 IEEE Symposium on Security and Privacy (S&P)*. IEEE, 636–654.
[12] Earlence Fernandes, Justin Paupore, Amir Rahmati, Daniel Simionato, Mauro Conti, and Atul Prakash. 2016. FlowFence: Practical Data Protection for Emerging IoT Application Frameworks. In *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016.* 531–548.
[13] Jun Han, Albert Jin Chung, Manal Kumar Sinha, Madhumitha Harishankar, Shijia Pan, Hae Young Noh, Pei Zhang, and Patrick Tague. 2018. Do you feel what I hear? Enabling autonomous IoT device pairing using different sensor types. In *2018 IEEE Symposium on Security and Privacy (S&P)*. IEEE, 0.
[14] Grant Ho, Derek Leung, Pratyush Mishra, Ashkan Hosseini, Dawn Song, and David Wagner. 2016. Smart locks: Lessons for securing commodity internet of things devices. In *Proceedings of the 11th ACM on Asia conference on computer and communications security (AsiaCCS)*. ACM, 461–472.
[15] Apple Inc. 2019. Apple HomeKit: Your home at your command. https://www.apple.com/uk/ios/home/ [Online; accessed 12-May-2019].
[16] SmartThings Inc. 2019. Samsung SmartThings. https://www.smartthings.com/ [Online; accessed 12-May-2019].
[17] Yunhan Jack Jia, Qi Alfred Chen, Shiqi Wang, Amir Rahmati, Earlence Fernandes, Z Morley Mao, and Atul Prakash. 2017. ContexIoT: Towards Providing Contextual Integrity to Appified IoT Platforms. In *Proceedings of the 24th Annual Network and Distributed System Security Symposium (NDSS)*.
[18] Constantinos Kolias, Georgios Kambourakis, Angelos Stavrou, and Jeffrey Voas. 2017. DDoS in the IoT: Mirai and Other Botnets. *Computer* 50, 7 (2017), 80–84.

[19] Gierad Laput, Yang Zhang, and Chris Harrison. 2017. Synthetic sensors: Towards general-purpose sensing. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. ACM, 3986–3999.
[20] Markus Miettinen, N Asokan, Thien Duc Nguyen, Ahmad-Reza Sadeghi, and Majid Sobhani. 2014. Context-based zero-interaction pairing and key evolution for advanced personal devices. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 880–891.
[21] Markus Miettinen, Samuel Marchal, Ibbad Hafeez, N Asokan, Ahmad-Reza Sadeghi, and Sasu Tarkoma. 2017. IoT Sentinel: Automated device-type identification for security enforcement in IoT. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2177–2184.
[22] Ben Nassi, Raz Ben-Netanel, Adi Shamir, and Yuval Elovici. 2018. Game of Drones - Detecting Streamed POI from Encrypted FPV Channel. *CoRR* abs/1801.03074 (2018). arXiv:1801.03074 http://arxiv.org/abs/1801.03074
[23] Dang Tu Nguyen, Chengyu Song, Zhiyun Qian, Srikanth V Krishnamurthy, Edward JM Colbert, and Patrick McDaniel. 2018. IotSan: fortifying the safety of IoT systems. In *Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*. ACM, 191–203.
[24] openHAB Foundation e.V. 2019. openHAB: empowering the smart home. https://www.openhab.org/ [Online; accessed 12-May-2019].
[25] Amir Rahmati, Earlence Fernandes, Kevin Eykholt, and Atul Prakash. 2018. Tyche: A risk-based permission model for smart homes. In *2018 IEEE Cybersecurity Development (SecDev)*. IEEE, 29–36.
[26] Eyal Ronen and Adi Shamir. 2016. Extended functionality attacks on IoT devices: The case of smart lights. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 3–12.
[27] Eyal Ronen, Adi Shamir, Achi-Or Weingarten, and Colin O'Flynn. 2017. IoT goes nuclear: Creating a ZigBee chain reaction. In *2017 IEEE Symposium on Security and Privacy (S&P)*. IEEE, 195–212.
[28] Brian C. Ross. 2014. Mutual information between discrete and continuous data sets. *PLoS ONE* 9, 2 (Feb. 2014), e87357. https://doi.org/10.1371/journal.pone.0087357
[29] Dominik Schürmann and Stephan Sigg. 2013. Secure communication based on ambient audio. *IEEE Transactions on mobile computing* 12, 2 (2013), 358–370.
[30] Sandra Siby, Rajib Ranjan Maiti, and Nils Ole Tippenhauer. 2017. IoTScanner: Detecting Privacy Threats in IoT Neighborhoods. In *Proceedings of the 3rd ACM International Workshop on IoT Privacy, Trust, and Security*. ACM, 23–30.
[31] Vijay Sivaraman, Dominic Chan, Dylan Earl, and Roksana Boreli. 2016. Smartphones attacking smart-homes. In *Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks*. ACM, 195–200.
[32] Yuan Tian, Nan Zhang, Yueh-Hsun Lin, XiaoFeng Wang, Blase Ur, Xianzheng Guo, and Patrick Tague. 2017. Smartauth: User-centered authorization for the internet of things. In *26th USENIX Security Symposium (USENIX Security 17)*. 361–378.
[33] Qi Wang, Wajih Ul Hassan, Adam Bates, and Carl Gunter. 2018. Fear and Logging in the Internet of Things. In *Proceedings of the 25th Annual Network and Distributed System Security Symposium (NDSS)*.
[34] Wei Zhang, Yan Meng, Yugeng Liu, Xiaokuan Zhang, Yinqian Zhang, and Haojin Zhu. 2018. HoMonit: Monitoring Smart Home Apps from Encrypted Traffic. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 1074–1088.