HAWatcher: Semantics-Aware Anomaly Detection for Appified Smart Homes

Chenglong Fu, *Temple University;* Qiang Zeng, *University of South Carolina;*
Xiaojiang Du, *Temple University*

https://www.usenix.org/conference/usenixsecurity21/presentation/fu-chenglong

This paper is included in the Proceedings of the
30th USENIX Security Symposium.

August 11–13, 2021

978-1-939133-24-3

Open access to the Proceedings of the
30th USENIX Security Symposium
is sponsored by USENIX.

HAWatcher: Semantics-Aware Anomaly Detection for Appified Smart Homes

Chenglong Fu
Temple University
chenglong.fu@temple.edu

Qiang Zeng
University of South Carolina
zeng1@cse.sc.edu

Xiaojiang Du
Temple University
xjdu@temple.edu

Abstract

As IoT devices are integrated via automation and coupled with the physical environment, anomalies in an appified smart home, whether due to attacks or device malfunctions, may lead to severe consequences. Prior works that utilize data mining techniques to detect anomalies suffer from high false alarm rates and missing many real anomalies. Our observation is that data mining-based approaches miss a large chunk of information about automation programs (also called *smart apps*) and devices. We propose *Home Automation Watcher* (HAWatcher), a semantics-aware anomaly detection system for appified smart homes. HAWatcher models a smart home's normal behaviors based on both event logs and semantics. Given a home, HAWatcher generates hypothetical correlations according to semantic information, such as apps, device types, relations and installation locations, and verifies them with event logs. The mined correlations are refined using correlations extracted from the installed smart apps. The refined correlations are used by a *Shadow Execution* engine to simulate the smart home's normal behaviors. During runtime, inconsistencies between devices' real-world states and simulated states are reported as anomalies. We evaluate our prototype on the SmartThings platform in four real-world testbeds and test it against totally 62 different anomaly cases. The results show that HAWatcher achieves high accuracy, significantly outperforming prior approaches.

1 Introduction

With the rapid growth of Internet of Things (IoT), smart homes gain booming popularity. As predicted by Gartner, there will be more than 500 IoT devices deployed in a typical household by 2022 [72]. IoT devices become increasingly integrated, thanks to IoT platforms such as SmartThings [21], Homekit [47], and OpenHAB [55]. These platforms provide interoperability among home IoT devices by different vendors, and allow them to work according to user-specified automation programs (also called *smart apps*).

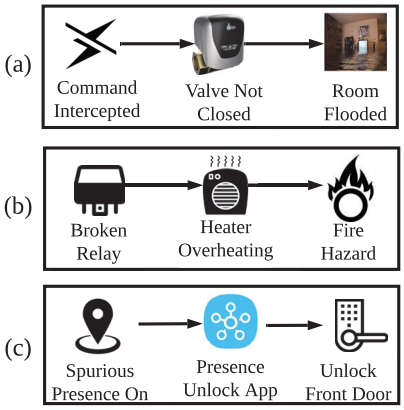

Figure 1: Examples of anomalies in a smart home.

Despite advances in appified smart home, there are growing concerns about its safety and security [41]. First, IoT devices make it possible for cyber-space attacks to be extended to the physical world. As shown in Figure 1(a), the command of "*close the valve*" is maliciously intercepted, which may cause room flooding. Second, very often a device malfunction is hardly noticeable until certain consequences arise. As shown in Figure 1(b), an electronic heater controlled by a smart app *"It's too cold"* [15] could result in fires because of a broken relay (an electronically operated switch), which prevents the plug from shutting the power for the heater. Third, as IoT devices are chained together via automation [28,29,39], abnormal behaviors of one device might trigger undesired actions of another, which further exaggerates the impact of anomalies. As shown in Figure 1(c), a smart lock that automatically unlocks upon the resident's presence is unlocked due to a fake event of the presence sensor.

To address these concerns, many anomaly detection systems [30,35,54,56,60,68,76] utilize data mining techniques to profile the system's normal behaviors and report events that deviate from profiles as anomalies. However, these works usually take event logs as inputs without fully considering each event's semantics, which actually may be acquired from smart apps, device types, and device functionalities. The lim-

itations are threefold. First, the logic of some smart apps is too complex to be mined accurately, causing false negatives and positives. For example, the event pattern introduced by the smart app logic "*Turn off a smart plug 30 minutes after two motion sensors in the living room are both motionless*" is difficult to be mined considering the 'AND' logic between two motion sensors and the 30 minutes action delay. As a result, an anomaly "*the smart plug fails to turn off*" may not be detected. Second, the learning results are typically difficult to interpret; thus, they can hardly be explained and often confuse users. Third, the learning results cannot be updated quickly when smart apps or configuration changes. A long re-training process is then needed to adapt to the changes and many false alarms arise before the re-training is done.

Intuitively, incorporating semantic information, such as automation logic, device types, relations and installation locations, can help improve the accuracy of anomaly detection. However, there are a number of challenges to overcome in order to realize this idea: 1) Standard data mining methods take event logs as inputs; however, it is unknown how to represent the diverse semantic information in the form of event logs. 2) System behavior patterns derived from smart apps and those mined from events logs may conflict. It is challenging to identify and resolve these conflicts. 3) When smart apps change, there are no effective methods to update the system profiling accordingly.

To fill the gap, we present *Home Automation Watcher* (HAWatcher), a novel anomaly detection system for appified home automation systems. We propose a semantics-assisted mining method that exploits diverse semantic information to construct *hypothetical correlations* (where a *correlation* describes how a device state or event correlates with another), and use event logs as evidence to verify them. Second, as the correlations are explainable according to the semantics, they can be easily refined to resolve conflicts with smart apps. Third, still thanks to explainability, they can be updated conveniently according to smart app changes. The correlations are then used by our *shadow execution* module to simulate normal behaviors in the *virtual world*. The simulated states are compared to those in the *real world* through both contextual checking and consequential checking, and inconsistencies during comparison are reported as anomalies. We make the following contributions.

- We propose a novel anomaly detection solution for appified smart homes. It meets the emerging need of detecting anomalies caused by IoT malfunctions or attacks.
- We propose a semantics-assisted mining method, which infuses various semantic information (smart apps, configuration, device types, installation locations) into the mining process. An NLP-based approach is developed to describe device relations for generating hypothetical correlations. The mined correlations are *explainable*, and can be refined easily to resolve conflicts with smart apps and updated conveniently when apps change.
- We propose the notion of *shadow execution* for smart homes, which simulates the normal behaviors of a home according to the learned correlations and detects anomalies at a fine granularity, i.e., *IoT events*.
- We implement a prototype HAWatcher and evaluate it on four real-world testbeds. HAWatcher reaches a high precision of 97.83% and a recall of 94.12%, significantly outperforming prior approaches.

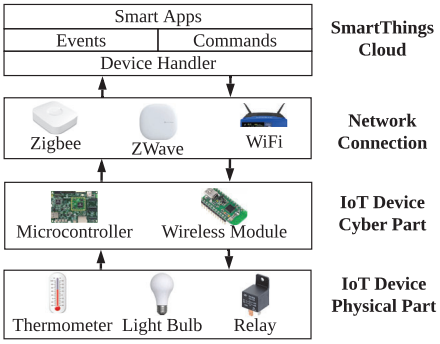

Figure 2: The SmartThings architecture.

The rest of the paper is organized as follows. In Section 2, we describe background about appified smart homes. In Section 3, we survey IoT device anomalies and present the threat model. In Section 4, we describe three correlation channels and the representation of correlations. We present the design details in Section 5. The evaluation is presented in Section 6. We discuss related work in Section 7, and limitations and future work in Section 8. The paper is concluded in Section 9.

2 Background: Appified Smart Homes

IoT devices in smart homes have become increasingly integrated via IoT platforms for rich automation. IoT integration platforms, such as SmartThings, Amazon Alexa, and OpenHAB, support trigger-action automation programs. On these platforms, despite the huge number of IoT devices, they are abstracted into a small number of *abstract devices*. For example, a smart light, regardless of its brand, shape, size, and wireless technology, is abstracted into the same abstract device, *light*. Each abstract device has its associated *events* and *commands*. Device vendors can have their products support integration by realizing the events and commands.

We choose SmartThings [21] as an example IoT integration platform to present our design, as SmartThings is one of the leading platforms and supports sophisticated automation logic. Other integration platforms, such as Amazon Alexa, have similar structures. As illustrated in Figure 2, a typical SmartThings deployment has a cloud-centric architecture of four layers. On the top is the SmartThings cloud, where smart apps run and interact with abstracted capabilities. The cloud

communicates with IoT devices through the network connection layer that uses various communication techniques such as WiFi, Zigbee, and ZWave. An IoT devices can be partitioned into the cyber part and the physical part. The cyber part manages interfaces for humans and bridges the communication between the cloud and the physical part, and the latter fulfills its functions in the physical world. Taking the Philips' Hue smart light bulb as an example, the physical part is the LED light bulb and the cyber part is the embedded micro-controller with a built-in wireless component.

Next, we describe some terms used in SmartThings. A device has one or multiple *capabilities*, each categorized as an *actuator* or *sensor*. Each capability defines one or more attributes. For example, a smart plug device has an attribute "*switch*" and, optionally, an attribute "*power*." Each attribute's *state* (i.e., value) is stored on the cloud and updated due to events sent from the IoT device. For example, the SmartThings multipurpose sensor has a capability *contact sensor*, whose attribute "*contact*" changes from "*open*" to "*closed*" when SmartThings receives an event of "*contact closed*" from the sensor. In addition, the state of an actuator's attribute is updated due to a *feedback event*, which is sent by the device after a command is executed by the actuator.

3 Motivation, Goals and Threat Model

IoT devices are notorious for their unreliability and insecurity [25, 40, 46]. Numerous anomalies in appified homes have been reported by users [4]. Below, we first discuss anomalies due to IoT device malfunctions and attacks as the motivation, and then present our goals and threat model.

3.1 IoT Device Malfunctions

We survey real-world anomalies frequently reported in the SmartThings user forum [4]. IoT devices interact with the IoT platform via events and commands; thus, we categorize malfunctions according to problematic events and commands.

Faulty Events. Faulty events refer to incorrect values reported by IoT devices. They can be caused by sensor defects or physical interference, such as mysterious door-knocking events [3] and motion events [9, 17, 46]. Faulty events may incorrectly trigger actuator actions and cause user confusions.

Ghost Commands. They are widely discussed in SmartThings' user forum, dubbed '*poltergeists*' [6, 12, 13]. For example, a smart plug was turned on itself at night, which overheated the connected waffle maker and electrical grill [5]. Users frequently reported their lights were turned on during midnight mysteriously [13].

Event Losses (or Large Delays). They refer to events that fail to be reported to the IoT cloud (in a timely manner). For example, mobile phone presence sensors were reported to suffer from a large delay on status update [8], which was confirmed by SmartThings [20]. Event losses may prevent the execution of related automation and leave the home in risky states. For example, the loss of a *presence-off* event could leave the door unlocked after the resident leaves home.

Command Failures. They correspond to commands issued by the IoT platforms that fail to be executed by the target devices. Command failures may be caused by malfunctions of a cyber part or physical part. (1) ***Cyber-part*** malfunctions that cause commands to fail to execute, such as system crashes and unstable network connections, are considered in our work. For example, the TP-Link smart plug often goes irresponsive [11]. (2) A ***physical-part*** malfunction is equivalent to a malfunction in a traditional (i.e., non-smart) device. For example, a broken electrical relay inside a smart plug can prevent the plug from cutting off the power supply [18], although from the perspective of the IoT platform, the plug has been turned off.

3.2 Attacks on IoT Devices

We survey the recent work on attacks against IoT devices, and find HAWatcher has the potential to detect the following five different types of attacks.

Fake Events. They are events maliciously injected by attackers. Fake events [80] may cause severe consequences by triggering actuator's actions. As illustrated in Figure 1(c), a fake presence-on event can unlock the door.

Fake Commands. An attacker may inject fake commands to IoT devices. For example, Sonos smart speaker [52] and WeMo Smart switch [62] accept commands from the local network without authenticating their origins [58, 70].

Event Interceptions. Events can be intercepted and discarded by attackers. E.g., the home security system can be muted by intercepting the window and door sensors' wireless connections to stop them from sending sensor events [66].

Command Interceptions. Similar to event interceptions, an attacker can also intercept a command and prevents it from being delivered to the device [43].

Compromised Devices. An attacker can compromise an IoT device and, at least, launch the following attacks. (1) ***Stealthy Commands***. The attacker can control the device to execute commands [65] and, to keep stealthy, stops the corresponding feedback events from being sent out.[1] (2) ***Denial of Executions (DoE).*** When a legitimate command is sent to the device, it does not execute the command but sends back a feedback event reporting the command has been executed.

3.3 Goals and Threat Model

We aim to detect both IoT device malfunctions described in Section 3.1 and attacks in Section 3.2. We clarify that HAWatcher can only detect attacks that violate correlations. Attackers who have knowledge of the correlations may construct attacks that do not violate any correlations and thus evade our detection, which is discussed in Section 8.

[1] If feedback events are not muted, it is much like a Fake Command.

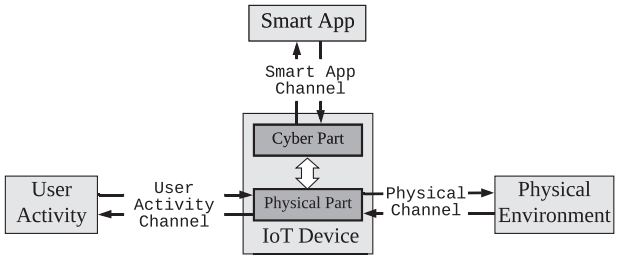

Figure 3: Correlation channels.

We assume the IoT platform is not compromised. Like other anomaly detection work [35,51,76], we assume there are no or very few anomalies during training. We assume there are no malicious or conflicting rules in the installed smart apps; how to detect malicious logic [71] and conflicting rules [28,34] are two separate research problems, and there are existing solutions to them [28,71], including our prior work [33,34]. Gartner predicts that a typical household could have more than 500 IoT devices by 2022 [72]. Given the dense deployment in the near future, we exploit scenarios where an IoT device has one or more other devices nearby to interact with, and propose to leverage them to detect a device's anomalous physical behaviors. We discuss the case of no interactive devices nearby in Section 8. Jamming that blocks communications reporting IoT events can be easily detected due to session timeout or missing sequence numbers; we thus do not further discuss it.

4 Correlations

Devices deployed in the same home may correlate in the form of co-present or temporally related events [35,39,45,68]. These correlations can be attributed to the execution of smart apps [29], physical interactions [39] or users' activities [45]. As shown in Figure 3, we investigate the causes of these correlations and categorize them into three channels below.

4.1 Correlation Channels

Smart App Channel. Smart apps not only directly cause correlations between triggers and actions as programmed, but also imply some extra correlations that should be considered. For example, the smart App "*light follows me*" [2] leads to the correlation between the motion sensor and the light, and also implies a possible correlation worth verification, that is, "*if the light is turned on, then the motion should be in the active state*". The implied correlation is true if the light is exclusively turned on by the smart app.

Physical Channel. Two devices can correlate via a certain physical property. First, an actuator device's action can change a physical property, which is captured by nearby sensor devices observing that property. For example, a smart light's action can affect an illuminance sensor nearby. Second, different sensor devices can be affected by the same physical event and generate temporally correlated IoT events. For instance, opening a door inevitably involves the door's movement, which could be captured by both a contact sensor and an acceleration sensor installed on the door and results in two consecutive events. With increasing types of IoT devices deployed, physical-channel correlations can be pervasively observed on many physical properties, such as illuminance, power, sound, and temperature [39].

User Activity Channel. While user activities impose changes on devices, device states also reflect user activities. Thus, the user activity channel causes correlations between devices. For example, a TV being turned on typically implies that the user is nearby, which should be captured by the motion sensor. When a user returns home, there should be consecutive events, such as "*presence on*" showing the user's proximity and "*contact-sensor open*" for door opening.

4.2 Representation of Correlations

An *event* reporting that the device A's attribute α should be changed to the value a is denoted as $\mathcal{E}_a^{\alpha(A)}$, while a *state* which indicates that the device B's attribute β has the value b is denoted as $\mathcal{S}_b^{\beta(B)}$.[2] We define two types of correlations.

- The *event-to-event (e2e)* correlation. It means that one event should be followed by (denoted as $\rightarrow$) another. For example, given a motion sensor A and a light B, the e2e correlation $\langle \mathcal{E}_{active}^{motion(A)} \rightarrow \mathcal{E}_{on}^{switch(B)} \rangle$ means the event $\mathcal{E}_{active}^{motion(A)}$ should be followed by the event $\mathcal{E}_{on}^{switch(B)}$.
- The *event-to-state (e2s)* correlation. It means that one event arising implies (denoted as $\rightsquigarrow$) a state is true. For example, $\langle \mathcal{E}_{high}^{power(plug)} \rightsquigarrow \mathcal{S}_{on}^{switch(heater)} \rangle$ means that, when the event $\mathcal{E}_{high}^{power(plug)}$ arises, the state $\mathcal{S}_{on}^{switch(heater)}$ should be true.

For the representation of a correlation involving conditions, its anterior event is combined with the conditions using the "$\wedge$" symbol. For example, $\langle \mathcal{E}_{active}^{Motion} \wedge \mathcal{S}_{present}^{Presence} \rightarrow \mathcal{E}_{on}^{switch(Light)} \rangle$ means the event $\mathcal{E}_{active}^{Motion}$, if the condition $\mathcal{S}_{present}^{Presence}$ is true, should be followed by $\mathcal{E}_{on}^{switch(Light)}$.

We show in Section 5 that the two types of correlations, despite their simplicity, are very effective in capturing rich semantic information and modeling the relations of devices that correlate via different channels.

5 HAWatcher Design and Implementation

We first introduce the workflow of anomaly detection (Section 5.1), and then describe the major modules in HAWatcher, as shown in Figure 4: 1) Semantic Analysis (Section 5.2), 2) Correlation Mining (Section 5.3), 3) Correlation Refining (Section 5.4), and 4) Anomaly Detection (Section 5.5).

[2] For simplicity of description, without causing confusion we sometimes omit the device IDs and use the simplified notations $\mathcal{E}_a^{\alpha}$ and $\mathcal{S}_b^{\beta}$.

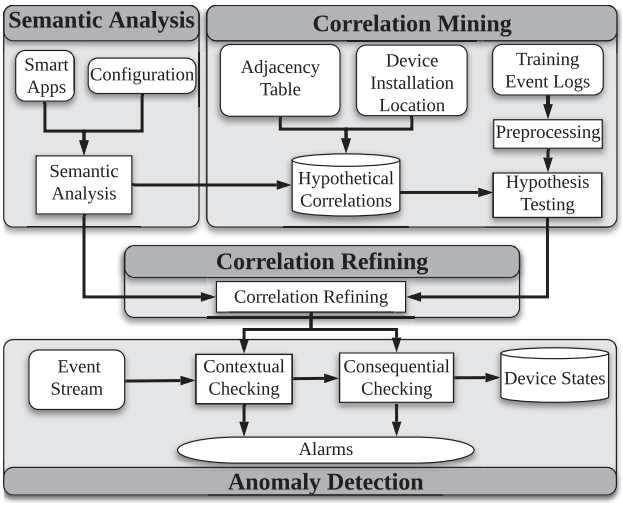

Figure 4: Architecture of HAWatcher.

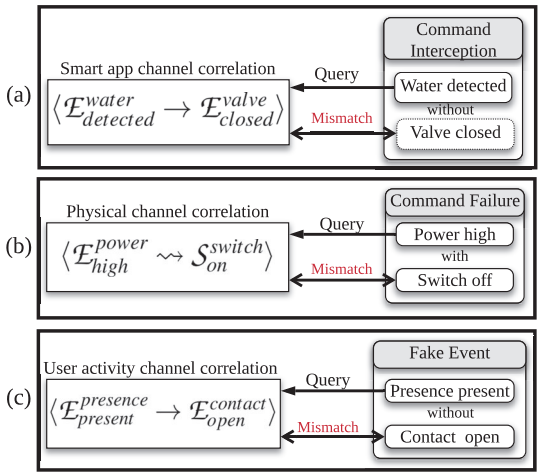

Figure 5: Detecting anomalies depicted in Figure 1.

```
def installed() {
    subscribe(lightSensor, "illuminance", illuminanceHandler)
}

def updated() {
    unsubscribe()
    subscribe(lightSensor, "illuminance", illuminanceHandler)
}

def illuminanceHandler(evt) {
    if (evt.integerValue < 30)
        lights.on()
    else if (evt.integerValue > 50)
        lights.off()
}
```

Figure 6: Code snippet of the app *LightUpTheNight*.

5.1 Workflow of Anomaly Detection

The Anomaly Detection module runs parallel with the appified home automation, and checks the events received from IoT devices against the learned correlations to detect anomalies. Figure 5 illustrates how this module detects anomalies, using anomalies depicted in Figure 1 as examples.

In case (a), the smart app automatically shuts the valve when water is detected. By applying semantic analysis to the app, HAWatcher extracts an e2e correlation $\langle \mathcal{E}^{water}_{detected} \rightarrow \mathcal{E}^{valve}_{closed} \rangle$. Since attackers intentionally intercept the command "*close the valve*" towards the valve, there is no feedback event $\mathcal{E}^{valve}_{closed}$, which contradicts the correlation. Furthermore, if it is a Command Failure caused by the valve's cyber-part malfunction, HAWatcher can detect it the same way.

In case (b), the hypothetical e2s correlation $\langle \mathcal{E}^{power}_{high} \rightsquigarrow \mathcal{S}^{switch}_{on} \rangle$ is first proposed based on the physical channel and then gets confirmed using the training event logs. After a turning-off command is sent to the plug and executed by its cyber part (hence, its *Switch=off*), however, due to its broken relay, the plug still supplies power and thus the power meter reports events of high power usage, which violates the aforementioned correlation and triggers an alarm.

In case (c), as the resident does not actually return home, there is no event $\mathcal{E}^{contact}_{open}$ that follows the fake event $\mathcal{E}^{presence}_{present}$. This deviates from the user activity channel correlation $\langle \mathcal{E}^{presence}_{present} \rightarrow \mathcal{E}^{contact}_{open} \rangle$ and is thus reported as an anomaly.

5.2 Semantic Analysis

The Semantic Analysis module executes two steps: (1) extract semantics from smart apps and their configuration, such as the temperature threshold for turning on AC and which IoT devices are bound to which app, and (2) convert the semantics to correlations.

Semantic analysis has been used to detect malicious or risky smart apps as in [41, 50, 79]. We use the method described in our prior work [33, 34] to extract semantics in Step (1). It applies symbolic execution to the Intermediate Representation of apps and captures the configuration information, achieving precise semantics extraction. The extracted semantics of each app is represented as one or more *rules*, each in the form of a tuple *trigger(T)-condition(C)-action(A)*, which means that "if T occurs, when C is true, execute A."

Step (2), which converts rules to correlations, is straightforward. Assuming T is reflected by the event E_1, and E_2 is the feedback event due to executing A, the rule above is converted to a correlation $\langle E_1 \wedge C \rightarrow E_2 \rangle$.

Taking a SmartThings official app *LightUpTheNight* [16] shown in Figure 6 as an example, the Semantic Analysis module converts it into two e2e correlations: $\langle \mathcal{E}^{Illuminance}_{<30} \rightarrow \mathcal{E}^{Light}_{on} \rangle$ and $\langle \mathcal{E}^{Illuminance}_{>50} \rightarrow \mathcal{E}^{Light}_{off} \rangle$. Here, note that the condition ("*Illuminance < 30*" or "*Illuminance >50*") and the trigger event in each rule refer to the same attribute of the same device; we thus merge the trigger and the condition to derive a concise representation of the trigger events.

Moreover, as described in Section 4.1, given an e2e correlation $\langle \mathcal{E}^{\alpha(A)}_{a} \rightarrow \mathcal{E}^{\beta(B)}_{b} \rangle$ extracted from the smart app, we further propose a hypothetical e2s correlation $\langle \mathcal{E}^{\beta(B)}_{b} \rightsquigarrow \mathcal{S}^{\alpha(A)}_{a} \rangle$, which means that the event $\mathcal{E}^{\beta(B)}_{b}$ only arises when $\mathcal{S}^{\alpha(A)}_{a}$ is

true. Such hypothetical e2s correlations are *not* necessarily true, and have to be verified using event logs (Section 5.3).

5.3 Correlation Mining

While there exist many pattern mining methods, few achieve both good usability and high accuracy in the context of appified home automation. Supervised mining methods [51, 77] are more accurate but require well annotated datasets or users' interventions. Unsupervised methods [31, 35, 60, 68] can be applied to unannotated data, but are less accurate.

Instead of relying on annotated datasets, we propose a semantic-based mining method. Semantic information includes devices' types and installation locations, which can be obtained from home automation platforms. Based on this information, HAWatcher proposes hypothetical correlations (in addition to those e2s correlations from smart apps) corresponding to physical channels and user activity channels. Each hypothetical correlation is then verified independently. Like other anomaly detection works [35, 51, 76], we assume there are no or very few anomalies during the training phase.

5.3.1 Prepossessing Event Logs

Prepossessing of event logs is necessary for two reasons: 1) Raw event logs are noisy with repetitive sensor readings. For example, some power meters periodically report similar (but slightly fluctuating) readings. 2) Devices' numeric readings cannot be incorporated into logical calculations. We thus design a preprocessing scheme for redundancy removal and numeric-to-binary conversion.

For each device that generates numeric readings, we add up its readings from the entire training dataset and calculate its mean μ and standard deviation σ. Readings that fall outside the range $[\mu - 3\sigma, \mu + 3\sigma]$ are excluded as extreme values (i.e., the three-sigma rule [64]).[3] Then, we apply the Jenks natural breaks classification algorithm [49][4] to the remaining readings and classify them as either '*low*' or '*high*'. Next, for each device's given attribute, we traverse the events and remove those that do not change the state (e.g., consecutive $\mathcal{E}_{high}^{Illuminance}$). Now, each two temporally adjacent events about the same attribute of a device have opposite values.

5.3.2 Hypothetical Correlation Generation

Besides those generated from the smart app channel, hypothetical correlations can be generated from the physical and user activity channels with other semantic information, such as device attributes and relations between attributes. We first utilize the semantic information to construct a table marking correlated attribute pairs; then, we fill each pair with devices that have matching attributes to generate hypothetical correlations.

[3] Event exclusion is for training only; the anomaly detection module does not eliminate events.

[4] Jenks natural breaks algorithm and K-means algorithm give the same results for one-dimension data [38]

Table 1: Part of the adjacency table. A cell marked with ✓means the corresponding attribute in the column may correlate with the one in the row head. The full table of 73*73 is in our technical report [44]

	Acceleration	CarbonDioxide	Contact	Illuminance	Motion	Power	Presence	Humidity	Sound	Button	Switch
Acceleration			✓		✓		✓		✓		✓
CarbonDioxide					✓		✓				✓
Contact	✓				✓		✓		✓		✓
Illuminance					✓		✓				✓
Motion	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓
Power					✓		✓				✓
Presence	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓
Humidity					✓		✓				✓
Sound	✓		✓		✓		✓				✓
Button					✓		✓				✓
Switch	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	

For physical channel correlations, we consider seven *physical properties* that are related to many smart home IoT devices: illuminance, sound, temperature, humidity, vibration, power, and air quality. To determine whether two IoT device attributes may relate via a physical property, we develop an NLP (Natural Language Processing) based approach. Specifically, for each attribute of an abstract IoT device, we obtain its description from the SmartThings' developer website [19] and parse it into a list of separate words. To objectively evaluate the relatedness between an attribute and a physical property, we use Google's pre-trained word2vec model [59] to calculate the semantic similarity scores between each word in the list and the physical property, and use the highest score as the *relatedness score* between the physical property and the attribute. For each physical property, we select the top ten attributes with the highest scores, which are considered mutually correlated via that physical property.

This way, we are able to find all correlated attribute pairs and mark them in an *adjacency table*, part of which is shown in Table 1. As SmartThings stipulates 73 attributes [19], the table is 73*73. A cell with ✓means that the attributes in its row head and column head correlate.

While most of the cells are automatically generated, an exception is the *switch* attribute: as all actuator devices have the switch attribute, we mark it as correlated with all other attributes. For user activity channel correlations, we use *presence* and *motion* as the two special attributes that directly reflect users' activities. As a user's activity may affect all the attributes, in the adjacency table we mark *presence* and *motion* as correlated with all other attributes.

For a specific smart home, all attributes of the installed devices are checked against this adjacency table to find pairs that may correlate. Given a pair of correlated attributes α and β in the adjacency table, the device A with the attribute α, and B with β, we generate four hypothetical e2e correlations $\langle \mathcal{E}_a^{\alpha(A)} \rightarrow \mathcal{E}_b^{\beta(B)} \rangle$, $\langle \mathcal{E}_{a'}^{\alpha(A)} \rightarrow \mathcal{E}_b^{\beta(B)} \rangle$, $\langle \mathcal{E}_a^{\alpha(A)} \rightarrow$

$\mathcal{E}_{b'}^{\beta(B)}\rangle$, $\langle \mathcal{E}_{a'}^{\alpha(A)} \rightarrow \mathcal{E}_{b'}^{\beta(B)}\rangle$, and four e2s ones ($\langle \mathcal{E}_{a}^{\alpha(A)} \rightarrow \mathcal{S}_{b}^{\beta(B)}\rangle$, $\langle \mathcal{E}_{a'}^{\alpha(A)} \rightarrow \mathcal{S}_{b}^{\beta(B)}\rangle$, $\langle \mathcal{E}_{a}^{\alpha(A)} \rightarrow \mathcal{S}_{b'}^{\beta(B)}\rangle$, $\langle \mathcal{E}_{a'}^{\alpha(A)} \rightarrow \mathcal{S}_{b'}^{\beta(B)}\rangle$, where a and a' (b and b', resp.) are values of the attribute α (β, resp.) after numeric-to-binary conversion; symmetrically, we generate another eight hypothetical correlations with the events of B as anteriors.

Moreover, we propose to combine semantics from smart apps with semantics from the adjacency table. The intuition behind the combination is that when an action command in a smart app is executed, it usually imposes certain changes on one or more attributes. Given an e2e correlation containing a condition extracted from a smart app, we create a *virtual device*, which reports an event when both the trigger event arises and the condition is true. For instance, a virtual motion sensor is created according to the conditional trigger $\mathcal{E}_{active}^{Motion(M)} \wedge \mathcal{S}_{present}^{presence(PS)}$, which becomes active only when $\mathcal{E}_{active}^{Motion(M)}$ arises and PS is present. Next, the virtual device is used, just like the corresponding real device, to generate hypothetical correlations according to the adjacency table.

Our current prototype only considers devices installed in the same room for generating hypothetical correlations. While this can be relaxed by considering any two devices in the home, our current implementation makes a trade-off between the comprehensiveness of hypothetical correlations and the meaningfulness of the mined correlations.

5.3.3 Hypothesis Testing

It is worth emphasizing that hypothetical correlations are not necessarily true. That is why we need hypothesis testing, the process of verifying hypothetical correlations using event logs. Given a hypothetical correlation, we traverse event logs to find all events that match its anterior, and take each of them as a testing case. Then, we check whether the hypothetical correlation's posterior event or state is consistent with the physical ground truth as recorded in event logs. For example, an event instance of $\mathcal{E}_{active}^{Motion}$ constitutes a testing case for the hypothetical correlation $\langle \mathcal{E}_{active}^{Motion} \rightarrow \mathcal{E}_{on}^{switch(Light)}\rangle$. This case is counted as a success if $\mathcal{E}_{on}^{switch(Light)}$ occurs within a short duration d after $\mathcal{E}_{active}^{Motion}$. In our implementation, d = 60s, which is long enough to wait for the feedback event to arrive but not too long as to accept an event not related to $\mathcal{E}_{active}^{Motion}$. Note the scheduling granularity of SmartThings is at per-minute level [1].

Checking these testing cases can be considered as a sequence of independent Bernoulli trails. We use the one-tail test [42] to evaluate each hypothetical correlation's correctness. For a given correlation, we set the alternative hypothesis H_α as "*the correlation succeeds with a probability higher than* P_0". Correspondingly, the null hypothesis H_0 is "*the correlation succeeds with a probability no higher than* P_0". We choose the 95% fiducial probability as in common practices [27], which means that the correlation can only be accepted if the null hypothesis's p-value is smaller than 5%.

5.4 Correlation Refining

The accepted hypothetical correlations should not be used directly for two reasons. First, conditions of smart apps may be overlooked if they remain unchanged during training. For instance, assume there is a smart app that, upon the front door opening, turns on the porch light after sunset. If the residents always come back home after sunset, the inaccurate correlation $\langle \mathcal{E}_{open}^{contact} \rightarrow \mathcal{E}_{on}^{switch(PorchLight)}\rangle$ could be accepted by hypothesis testing and cause false alarms of "*porch light not turned on*" when the residents return before sunset. Second, when apps change, accepted hypothetical correlations may become outdated and contradict with the e2e correlations newly derived from apps. This can also cause false alarms, as confirmed by our experiments (Section 6.5).

We thus propose to refine mined correlations using e2e correlations extracted from smart apps, and launch the refining process whenever smart app changes or there are hypothetical correlations accepted by hypothesis testing. We first define the *cover* relation between two correlations: an e2e correlation $\mathbb{C}_s = \langle \mathcal{E}_a^{\alpha(A)} \rightarrow \mathcal{E}_b^{\beta(B)}\rangle$ extracted from a smart app *covers* a correlation $\mathbb{C}_h = \langle \mathcal{E}_c^{\gamma(C)} \rightarrow \mathcal{E}_d^{\delta(D)}\rangle$ that passes hypothesis testing if they meet two conditions: 1) they have the same posterior event (i.e., $\mathcal{E}_b^{\beta(B)} = \mathcal{E}_d^{\delta(D)}$); and 2) $\mathcal{E}_a^{\alpha(A)}$ (logically) implies $\mathcal{E}_c^{\gamma(C)}$ (i.e., $\mathcal{E}_a^{\alpha(A)} \Rightarrow \mathcal{E}_c^{\gamma(C)}$). If $\mathbb{C}_s$ covers $\mathbb{C}_h$, the latter is removed. In the example mentioned above, a smart app derived e2e correlation $\langle \mathcal{E}_{open}^{contact} \wedge \mathcal{S}_{sunset}^{location} \rightarrow \mathcal{E}_{on}^{switch(PorchLight)}\rangle$ covers the mined correlation $\langle \mathcal{E}_{open}^{contact} \rightarrow \mathcal{E}_{on}^{switch(PorchLight)}\rangle$ because they have the same posterior event and $(\mathcal{E}_{open}^{contact} \wedge \mathcal{S}_{sunset}^{location}) \Rightarrow \mathcal{E}_{open}^{contact}$; thus, the latter correlation is removed.

5.5 Anomaly Detection

SmartThings does not provide access to its internal content, such as device states. To overcome the barrier, we design a *shadow execution engine*, which subscribes to the events of the installed IoT devices. It keeps track of all devices' states and simulates a smart home's legitimate behaviors based on obtained correlations.

For each incoming event, the shadow execution engine performs the *Contextual* and *Consequential* checking successively. The contextual checking verifies whether the event occurs in a valid context specified in e2s correlations. After that, the consequential checking searches for its consequential events as predicted by e2e correlations.

Below, we use the same example correlation (between a motion sensor and a light) as in Section 4.2. When an event $\mathcal{E}_{active}^{Motion(A)}$ is received, the shadow execution engine first conducts the contextual checking. It traverses all e2s correlations and locates those with the event $\mathcal{E}_{active}^{Motion(A)}$ at their anterior places. Among the located e2s correlations, if any of them have states in their posterior places that are inconsistent

Table 2: Numbers of rooms, devices and apps in each testbed.

Testbed	#Rooms	#Devices	#Smart apps
1	5	23	17
2	4	19	11
3	1	6	7
4	1	6	4

with the real-world devices' states, an alarm is raised reporting the event $\mathcal{E}_{active}^{Motion(A)}$ as invalid. Otherwise, the event is accepted and the shadow execution engine changes its simulated motion sensor's state to "*active*" accordingly. Then, for each accepted event (motion *A* turns "*active*" in the example), the shadow execution engine performs the consequential checking. It searches all e2e correlations that have $\mathcal{E}_{active}^{Motion(A)}$ at their anterior places and caches events at their posterior places in a waiting list. If any event in the list is not received within 60 seconds (consistent with *d* in hypothesis testing), the shadow execution engine reports an anomaly of a missing event. Moreover, an event from a real device also induces an event from its derived *virtual device* (defined in Section 5.3.2) if the involved condition is true, and the event of the *virtual device* is handled in the same way as that from the real device through contextual and consequential checking.

6 Evaluation

We evaluate HAWatcher with datasets collected from 4 different real-world testbeds as shown in Figure 7. On each testbed, we spend three weeks collecting dataset for training and one week for testing. We apply collected correlations to each event from the testing datasets to evaluate HAWatcher's performance. We compare HAWatcher with other anomaly detectors. Here, we mainly present evaluation results of Testbed 1. The results of other testbeds are presented in Appendix A.2.

6.1 Experimental Setup

While there are several existing datasets from smart homes or home activity learning researches, such as [36,37], none of these are collected from appified home testbeds. In addition, these testbeds contain mainly sensor devices but very few actuator devices. These make them unsuitable for evaluating HAWatcher, which is designed to work with appified homes. Next, we describe how we set up our testbeds.

Testbeds and Participants. We deploy SmartThings systems in four homes and Table 2 lists their basic information. Testbeds 1 and 2 each have two residents, and testbeds 3 and 4 have one resident each. The six (6) participants consist of 5 graduate students and 1 undergraduate student including two females and four males. Two of them are members of our research lab and none paper authors. None of them had prior experience of using home automation systems. For each testbed, we let the resident(s) propose desired automation, which is fulfilled by us with off-the-shelf IoT devices and smart apps from the SmartThings official repository. We then give them sufficient time to get familiar with the installed home automation before starting data collection.

Table 3: IoT devices used in the four testbeds, their abbreviation labels, attributes and deployment information.

Abbr.	Device Name	Attributes	Deployment
M	*SmartThings Motion Sensor*	motion	on wall
MS	*Zooz 4-in-1 Sensor*	motion, illuminance, humidity	on wall
W	*SmartThings Waterleak Sensor*	water	on bathroom floor
C	*SmartThings Contact Sensor*	contact, acceleration	on doors
B	*SmartThings Button*	button	bedside
L	*SmartThings Light Bulb*	switch	as ceiling light, lamp
PS	*SmartThings Arrival sensor*	presence	in wallet
P	*SmarThings Smart Plug*	switch, power	to control fan, computer, and lamp
A	*Netatmo Air Station*	carbonDioxide, sound, humidity	on kitchen countertop
V	*ThreeReality Smart Switch*	switch	to control fan

Table 4: Automation rules used in Testbed 1.

Index	Smart app rules
R1	If M1(active) when Mode(home), then P3(on)
R2	If M2(active) when Mode(home), then P4(on)
R3	If MS1(active), then L1(on) and L2(on)
R4	If MS1(inactive) for 15 minutes, then L1(off) and L2(off)
R5	If MS2(active), then L3(on)
R6	If MS2(inactive) for 10 minutes, then L3(off)
R7	If MS3(active), then L4(on)
R8	If MS3(inactive) for 5 minutes, then L4(off)
R9	If MS4(active), then L5(on)
R10	If MS4(inactive) for 15 minutes, then L5(off)
R11	If B(pressed), then toggle P3 and P4
R12	If B(held), then turn off all L and P and Mode(night)
R13	If B(double pressed), turn on P3 and P4 and Mode(home)
R14	If A($CO_2 \geq 950$), then P2(on)
R15	If A($CO_2 \leq 950$), then P2(off)
R16	If PS1 and PS2 (away), then turn off all L and P and Mode(away)
R17	If PS1 or PS2 (present), then turn on L1, L2, and P1 and Mode(home)

Device Deployment. The device deployment is depicted in Figure 7. We deploy 10 different types of IoT devices as listed in Table 3, including their abbreviation labels. Note that the ThreeReality Smart Switch (denoted as **V**) can be attached to a wall switch to control traditional devices, such as lights and fans. The smart plug (denoted as **P**) can be used to control electrical appliances with power plugs; for example, in Testbed 1, **P1** and **P2** are connected to a TV and a fan, respectively, and **P3** and **P4** are connected to lamps.

Automation Rules. We extract automation rules from the installed smart apps in the form of "If *trigger* when *condition*, then *action*". The extracted rules of Testbed 1 are listed in Table 4 (rules of other testbeds are presented in Appendix A.1).

Ethical Concerns and Mitigation. We obtained the IRB

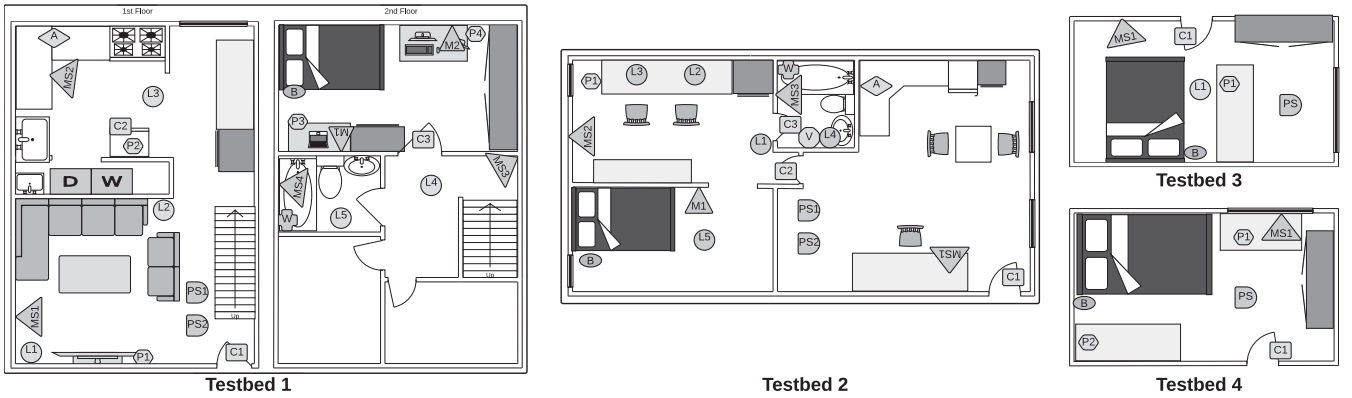

Figure 7: Floor plans of four testbeds and device deployment layouts (the device abbreviation labels are illustrated in Table 3).

approval for the study. All participants are fully aware of all the installed devices and apps. We do not use any sensitive devices such as cameras and microphones. The sound sensor of Device *A* in Table 3 only reports the sound level rather than the raw audio. All data is considered sensitive and personal identifiable information (PII) is removed right after collection for long-term storage. We store all the data in an encrypted hard drive mounted to our lab's server, which is only accessible to accounts of the paper authors.

For the purpose of testing, we need to inject anomalies (see Section 6.3). To avoid safety issues, the injected anomalies do not target any safety-sensitive devices, such as heaters. We notify participants of incoming testing one day ahead but do not disclose the details (e.g., device and time) of the anomaly cases. We also ask participants to keep their normal living habits and do not panic if they notice any anomalies. The purpose is to avoid their behavioral bias during testing. Details of the injected anomalies are presented to participants after the testing.

6.2 Training

Training HAWatcher. From Testbed 1, we generate 46 e2e correlations from the automation rules. In addition, we generate totally 2,398 hypothetical correlations, including 46 e2s correlations from the smart app channel, 544 from the physical channel, and 1,808 from the user activity channel. Then, the hypothetical correlations are checked using 22,655 events collected from the three weeks' training phase. In total, 146 correlations are accepted by hypothesis testing, and 130 remain after refining. On other three testbeds, the portion of smart app channel correlations are 32/109, 15/55, and 8/26, respectively. Table 5 lists a portion of the correlations after refining. Some correlations reveal interesting facts that are confirmed by the residents.

Observation 1: While *C1* and *C3* are both contact sensors, *C1* has one additional correlation $\mathbb{C}11 = \langle \mathcal{E}_{active}^{acceleration(C1)} \rightarrow \mathcal{E}_{closed}^{contact(C1)} \rangle$, which means the event $\mathcal{E}_{active}^{acceleration(C1)}$ should be followed by $\mathcal{E}_{closed}^{contact(C1)}$. This is because the front door (with *C1*) is typically closed right after being opened, while the bedroom door (with *C3*) does not have this pattern.

Observation 2: The e2s correlation $\mathbb{C}23$ means that *MS3*'s illuminance goes *high* only when *L4* is on. This is because there are no other light sources near *MS3*. Other illuminance sensors do not have such a correlation as the high illuminance value can be caused by multiple lights or natural lights.

Observation 3: Smart plugs *P2* and *P4* are to turn on/off a fan and a lamp, respectively. Whenever *P2* and *P4* are turned on, higher power use is observed (see e2e correlations $\mathbb{C}16$ and $\mathbb{C}10$ in Table 5). However, for *P1* that is connected to a TV, $\mathcal{E}_{on}^{switch(P1)}$ is not followed by a power-high event, as the TV needs to be further turned on manually by the residents.

Observation 4: Physical- and user activity-channel correlations cannot be obtained without mining, since they are not included in any smart apps. On the other hand, some correlations can be easily extracted from smart apps but difficult to mine. For example, correlations that involve delays are difficult to be mined accurately, but can be precisely derived from rules, such as **R4**, **R6**, **R8**, and **R10**.

Training Baseline Approaches. We select the Association Rule Mining (ARM) [24] and the One-class Support Vector Machine (OCSVM) [67] based detectors as two baseline approaches. We choose OCSVM because it is wiedly used for anomaly detection and trained with one class of input data, which is suitable for our training data containing no or few anomalies [53]. ARM is selected because it is a well-established method for mining correlations/rules, and HAWatcher is also based on correlation mining.

We perform ARM [24] on the same training dataset for comparison. Since ARM algorithms require transaction-form inputs, we segment the training dataset at places where the time interval between two consecutive events is longer than 60s (the same as the threshold d used for hypothesis testing). By using the library *pymining* [22], we mine 221 association rules with the confidence threshold of 0.95. Unlike our correlation mining method that covers various attributes and devices, rules produced by the association rule mining are dominated by motion sensors *MS3* and *MS4*. All the 221 rules have either *MS3* or *MS4*'s motion attributes in their conse-

Table 5: A portion of refined correlations acquired from Testbed 1.

ID	Correlation	ID	Correlation	ID	Correlation	ID	Correlation
$\mathbb{C}1$	$\langle \mathcal{E}_{low}^{illuminance(MS3)} \rightsquigarrow \mathcal{S}_{off}^{switch(L4)} \rangle$	$\mathbb{C}2$	$\langle \mathcal{E}_{active}^{motion(MS1)} \rightarrow \mathcal{E}_{on}^{switch(L1)} \rangle$	$\mathbb{C}3$	$\langle \mathcal{E}_{present}^{presence(PS1)} \rightarrow \mathcal{E}_{open}^{contact(C1)} \rangle$	$\mathbb{C}4$	$\langle \mathcal{E}_{present}^{presence(PS1)} \rightarrow \mathcal{E}_{closed}^{contact(C1)} \rangle$
$\mathbb{C}5$	$\langle \mathcal{E}_{present}^{presence(PS2)} \rightarrow \mathcal{E}_{open}^{contact(C1)} \rangle$	$\mathbb{C}6$	$\langle \mathcal{E}_{high}^{power(P2)} \rightsquigarrow \mathcal{S}_{on}^{switch(P2)} \rangle$	$\mathbb{C}7$	$\langle \mathcal{E}_{present}^{presence(PS2)} \rightarrow \mathcal{E}_{active}^{motion(MS1)} \rangle$	$\mathbb{C}8$	$\langle \mathcal{E}_{pushed}^{button(B)} \rightarrow \mathcal{E}_{active}^{motion(M1)} \rangle$
$\mathbb{C}9$	$\langle \mathcal{E}_{open}^{contact(C1)} \rightarrow \mathcal{E}_{active}^{acceleration(C1)} \rangle$	$\mathbb{C}10$	$\langle \mathcal{E}_{on}^{switch(P4)} \rightarrow \mathcal{E}_{high}^{power(P4)} \rangle$	$\mathbb{C}11$	$\langle \mathcal{E}_{active}^{acceleration(C1)} \rightarrow \mathcal{E}_{closed}^{contact(C1)} \rangle$	$\mathbb{C}12$	$\langle \mathcal{E}_{on}^{switch(L4)} \rightarrow \mathcal{E}_{high}^{illuminance(MS3)} \rangle$
$\mathbb{C}13$	$\langle \mathcal{E}_{off}^{switch(L4)} \rightarrow \mathcal{E}_{low}^{illuminance(MS3)} \rangle$	$\mathbb{C}14$	$\langle \mathcal{E}_{on}^{switch(L3)} \rightsquigarrow \mathcal{S}_{active}^{motion(MS2)} \rangle$	$\mathbb{C}15$	$\langle \mathcal{E}_{on}^{switch(L3)} \rightarrow \mathcal{E}_{high}^{illuminance(MS2)} \rangle$	$\mathbb{C}16$	$\langle \mathcal{E}_{on}^{switch(P2)} \rightarrow \mathcal{E}_{high}^{power(P2)} \rangle$
$\mathbb{C}17$	$\langle \mathcal{E}_{active}^{acceleration(C3)} \rightarrow \mathcal{E}_{active}^{motion(MS3)} \rangle$	$\mathbb{C}18$	$\langle \mathcal{E}_{closed}^{contact(C1)} \rightsquigarrow \mathcal{S}_{active}^{motion(MS1)} \rangle$	$\mathbb{C}19$	$\langle \mathcal{E}_{on}^{switch(L4)} \rightsquigarrow \mathcal{S}_{active}^{motion(MS3)} \rangle$	$\mathbb{C}20$	$\langle \mathcal{E}_{closed}^{contact(C1)} \rightsquigarrow \mathcal{S}_{active}^{acceleration(C1)} \rangle$
$\mathbb{C}21$	$\langle \mathcal{E}_{closed}^{contact(C3)} \rightsquigarrow \mathcal{S}_{active}^{acceleration(C3)} \rangle$	$\mathbb{C}22$	$\langle \mathcal{E}_{active}^{motion(MS3)} \rightarrow \mathcal{E}_{on}^{switch(L4)} \rangle$	$\mathbb{C}23$	$\langle \mathcal{E}_{high}^{illuminance(MS3)} \rightsquigarrow \mathcal{S}_{on}^{switch(L4)} \rangle$	$\mathbb{C}24$	$\langle \mathcal{E}_{low}^{illuminance(MS1)} \rightsquigarrow \mathcal{S}_{off}^{switch(L1)} \rangle$
$\mathbb{C}25$	$\langle \mathcal{E}_{present}^{presence(PS1)} \rightarrow \mathcal{E}_{active}^{motion(MS1)} \rangle$	$\mathbb{C}26$	$\langle \mathcal{E}_{active}^{motion(MS1)} \rightsquigarrow \mathcal{S}_{on}^{switch(P1)} \rangle$	$\mathbb{C}27$	$\langle \mathcal{E}_{active}^{acceleration(C1)} \rightsquigarrow \mathcal{S}_{open}^{contact(C1)} \rangle$	$\mathbb{C}28$	$\langle \mathcal{E}_{active}^{acceleration(C2)} \rightsquigarrow \mathcal{S}_{active}^{motion(MS2)} \rangle$
$\mathbb{C}29$	$\langle \mathcal{E}_{on}^{switch(L5)} \rightarrow \mathcal{E}_{high}^{illuminance(MS4)} \rangle$	$\mathbb{C}30$	$\langle \mathcal{E}_{on}^{switch(P2)} \rightsquigarrow \mathcal{S}_{>950}^{CO_2(A)} \rangle$	$\mathbb{C}31$	$\langle \mathcal{E}_{on}^{switch(P3)} \rightsquigarrow \mathcal{S}_{active}^{motion(M1)} \rangle$	$\mathbb{C}32$	$\langle \mathcal{E}_{high}^{power(P3)} \rightsquigarrow \mathcal{S}_{on}^{switch(P3)} \rangle$
$\mathbb{C}33$	$\langle \mathcal{E}_{open}^{contact(C2)} \rightsquigarrow \mathcal{S}_{active}^{motion(MS2)} \rangle$	$\mathbb{C}34$	$\langle \mathcal{E}_{>950}^{CO_2(A)} \rightarrow \mathcal{E}_{on}^{switch(P2)} \rangle$	$\mathbb{C}35$	$\langle \mathcal{E}_{high}^{CO_2(A)} \rightsquigarrow \mathcal{S}_{active}^{motion(MS2)} \rangle$	$\mathbb{C}36$	$\langle \mathcal{E}_{high}^{sound(A)} \rightsquigarrow \mathcal{S}_{active}^{motion(MS2)} \rangle$
$\mathbb{C}37$	$\langle \mathcal{E}_{open}^{contact(C1)} \rightsquigarrow \mathcal{S}_{present}^{presence(PS1)} \vee \mathcal{S}_{present}^{presence(PS2)} \rangle$			$\mathbb{C}38$	$\langle \mathcal{E}_{active}^{motion(M2)} \wedge \mathcal{S}_{home}^{mode} \rightarrow \mathcal{E}_{on}^{switch(P4)} \rangle$		

Table 6: Impact of Different Training-Phase Duration

Training phase (days)	Precision	Recall	# of false alarms	# of correlations
3	63.63%	78.69%	212	183
6	75.35%	85.78%	147	141
9	94.57%	94.12%	15	135
12	97.25%	94.12%	8	132
15	97.83%	94.12%	4	130
18	97.83%	94.12%	4	130
21	97.83%	94.12%	4	130

quent event set and 214 of them have that in their antecedent event set. There are 80 rules involving lights *L4* and *L5*, 32 with illuminance sensors in *MS3* and *MS4*, and 14 with the CO_2 sensor in *A*. Other attributes are not seen in any rules, as events involving them are overshadowed by those involving the four aforementioned attributes. In contrast, with our mining method, each attribute is involved in at least four (4) correlations and has an average of 10.5 correlations.

For the OCSVM-based detector, it takes a snapshot of all devices' states as a *frame* each time a new event arises and concatenates four consecutive frames as one input data *vector* [48]. We use the open source OCSVM implemetation in sklearn [63] and the default kernel (Radial Basis Function).

Impact of Training-Phase Duration We study the impact of the duration of the training phase on the performance of HAWatcher. As Testbed 1 is the most complex one among the four testbeds, we select it in this experiment. As illustrated in Table 6, we start from using the first three (3) days of data as a training dataset, and then use the first six (6) days by increasing three days of data, and so on until we use all the 21 days of data. With each of the seven (7) training datasets, we train a system and evaluate its performance using the fourth week of testing data.

Based on the study and the results shown in Table 6, we have the following observations. (1) Nine (9) days of training data is enough for HAWatcher to achieve the highest detection recall, but its number of false alarms has not reached the lowest, which means some false correlations are obtained. (2) For the first two training datasets, although they lead to more correlations than the subsequent ones, the overall quality of correlations is not high. The reason is that we use the one-tail test (Section 5.3.3), which has two impacts. On the one hand, even a very small number of abnormal behaviors in the small datasets will cause some true correlations to be rejected. On the other hand, due to the small amount of data, many false correlations are not rejected yet. (3) Starting from the dataset of 15 days, the performance (including the number of false alarms) does not change anymore, which means that amount of data is sufficient for the testbed. (4) Those true correlations which have been rejected in the small datasets are recovered in the larger datasets. This shows the robustness of the design of HAWatcher. Even if very few anomalies arise during the training phase, true correlations can survive given sufficient training data. (5) We examine the different sets of correlations mined based on different duration and find that some false correlations remain there until more data is available. For example, $\langle \mathcal{E}_{high}^{humidity(MS3)} \rightsquigarrow \mathcal{S}_{closed}^{contact(C3)} \rangle$ remains until behaviors that fail the correlation appear on Days 11 and 12.

6.3 Anomaly Generation

To evaluate HAWatcher, we simulate 24 cases of anomalies on Testbed 1 listed in Table 7 (totally 62 cases on the four testbeds). We follow two criteria to select anomaly cases: (1) the attacks are discussed in the literature about IoT attacks; and (2) the malfunctions are frequently discussed in the SmartThings community. To simulate an anomaly case, we either modify the testing event logs (collected in the fourth week) or interfere with the home automation, and the resulting logs are used for anomaly detection. For each case, multiple instances (see the "#inst." column) are injected.

If an attack has the same impact on the event logs as a malfunction, we group and simulate them as one case. Taking Case 1 as an example, we randomly inject a total of 50 motion events of MS1 into the testing event logs to simulate the effect of both Faulty Events (due to sensor malfunctions) and Fake Events (due to attacks).

Faulty/Fake Events. We simulate them by inserting events of devices, such as motion sensors [17], presence sensor [14], and contact sensors [3], as they are reportedly unreliable.

Table 7: HAWatcher's detection performance on Tesbed 1. "#inst." indicates the number of instances for one testing case. As switch is a common attribute for all actuators, we point out the specific appliance controlled by each switch after the colon.

Case	Type	Anomaly Description	Anomaly Creation Method	#inst.	Precision	Recall	Correlations Violated
1	Faulty/Fake Events	false motion(MS1) active	insert events into the dataset	50	97.77%	86.00%	ℂ26
2		false contact(C1) open		50	100.00%	100.00%	ℂ9
3		false acceleration(C1) active		50	97.87%	92.00%	ℂ27
4		false presence(PS1,PS2) present		50	96.15%	100.00%	ℂ3,ℂ5,ℂ25,ℂ7
5		false button(B) pushed		50	100.00%	100.00%	ℂ8
6	Event Losses/ Interceptions	missing motion(MS2) active	remove events from the dataset	57	100.00%	92.98%	ℂ28, ℂ35, ℂ36, ℂ14
7		missing motion(MS3) active		38	100.00%	100.00%	ℂ17
8		missing contact(C1) open		11	78.57%	100.00%	ℂ3, ℂ5, ℂ27
9		missing presence(PS1,PS2) present		9	77.78%	77.78%	ℂ37
10		missing illuminance(MS3) events		46	100.00%	43.47%	ℂ12,ℂ13
11	Ghost/Fake Commands	turn on switch(P2):fan	toggle from the *ghost* smart app	50	100.00%	100.00%	ℂ30
12		turn on switch(P3):lamp		50	100.00%	100.00%	ℂ31
13		turn on switch(L4):light		50	100.00%	100.00%	ℂ19
14	Stealthy Commands	stealthily turn on switch(P2):fan	toggle from the *ghost* smart app and remove feedback events	50	100.00%	100.00%	ℂ6
15		stealthily turn on switch(P3):lamp		50	100.00%	100.00%	ℂ32
16		stealthily turn on switch(L4):light		50	100.00%	100.00%	ℂ23
17	Command Failures (cyber)/ Command Interceptions	fail to turn on switch(L1):light	cut off devices' power supply	9	100.00%	100.00%	ℂ2
18		fail to turn on switch(L4):light		12	100.00%	100.00%	ℂ22
19		fail to turn on switch(P2):fan		10	100.00%	100.00%	ℂ34
20		fail to turn on switch(P4):lamp		53	100.00%	100.00%	ℂ38
21	Command Failures (physical)/ Denial of Executions	fail to turn on switch(L1):light	cover bulbs with paper	9	100.00%	66.67%	ℂ24
22		fail to turn on switch(L4):light		12	100.00%	100.00%	ℂ12, ℂ1
23		fail to turn on switch(P2):fan	unplug connected appliances	10	100.00%	100.00%	ℂ16
24		fail to turn on switch(P4):lamp		53	100.00%	100.00%	ℂ10
Avg	-	-	-	-	97.83%	94.12%	-

Event Losses/Interceptions. To simulate them, we randomly remove events of some devices from the testing event logs. We select various types of devices that users complain about event losses, such as presence sensors [20], contact sensors [23], and motion sensors [10].

Ghost/Fake Commands Both smart lights and plugs have been frequently reported by users for turning on/off unexpectedly [5, 6, 12]. We write a *ghost* smart app, which is not known by HAWatcher, and use the app randomly issue commands to turn on smart lights and plugs.

Stealthy Commands With compromised smart lights [65] and plugs [58], attackers can control them to make stealthy but hazardous actions. We simulate this type of attacks using the same method as ghost/fake commands but remove the feedback event of each fake command.

Command Failures (cyber)/Command Interceptions We simulate Command Failures (cyber-part malfunctions) and Command Interceptions on smart plugs [11] and smart lights [7]. We cut the power of target devices to make them irresponsive. For each target device, we conduct the experiment multiple times during one day.

Command Failures (physical)/Denial of Executions Command Failures (physical part malfunctions) and Denial of Executions are simulated on lights [65] and smart plugs [18]. We cover smart lights with a lightproof paper, and unplug appliances from smart plugs. The smart lights and plugs still respond to commands with feedback events, but those commands would not have any physical effect. For each case, we conduct the experiment multiple times during one day.

6.4 Performance of Anomaly Detection

We first evaluate HAWatcher's precision and recall in detecting anomalies, and compare them with two baseline detectors. We then measure the false alarm rate of HAWatcher.

Evaluation Metrics. Given an anomaly case (see Table 7), *precision* is the number of correctly detected instances of that case divided by the number of alarms reporting that anomaly case (i.e., ratio of true anomalies to alarms), *recall* is the number of correctly detected instances of that case divided by the number of injected instances of that case (i.e., percentage of anomalies that can be detected), and the false alarm rate is the number of false positives divided by the number of IoT events.

$$\begin{aligned} Precision &= \frac{True\ Positive}{True\ Positive + False\ Positive} \\ Recall &= \frac{True\ Positive}{True\ Positive + False\ Negative} \\ False\ Alarm\ Rate &= \frac{False\ Positive}{All\ Events} \end{aligned} \tag{1}$$

Detectors for Comparison. We compare the performance of HAWatcher with that of two baseline approaches described in Section 6.2, *ARM* and *OCSVM*. For the ARM-based detector, we segment the testing dataset as during the training phase, and check each segment against all mined rules to detect anomalies. For the OCSVM-based detector, as in [48], we take a snapshot of all devices' states as a *frame* each time a new event arises and concatenate four consecutive frames as one data *vector*, which is fed into the trained OCSVM for detecting anomalies.

In addition, to evaluate the effect of semantic analysis of smart apps and correlation mining each and also to measure

the benefit brought by the combination of the two, we build two variants of HAWatcher: *HAWatcher (Apps Only)*, which extracts correlations from smart apps only, and *HAWatcher (Mining Only)*, which mines correlations without using apps.

Detection Results of HAWatcher. As shown in Table 7, HAWatcher has an average detection precision of 97.83% and a recall of 94.12% across the 24 diverse anomaly cases. For 18 out of 24 cases, HAWatcher successfully detects all the instances. Below we describe some examples to illustrate how HAWatcher detects anomalies.

Detecting Case 7. Residents entering/leaving the bedroom open the door, which is installed with an acceleration sensor *C3*, and cause the motion-active event of *MS3*. However, as motion-active events of *MS3* are intercepted/lost, the user activity e2e correlation $\mathbb{C}17 = \langle \mathcal{E}_{active}^{acceleration(C3)} \rightarrow \mathcal{E}_{active}^{motion(MS3)} \rangle$ is violated and the anomaly is hence detected.

Detecting Case 11. Ghost/Fake Commands that try to turn on *P2* are detected due to a violation of the correlation $\mathbb{C}30 = \langle \mathcal{E}_{on}^{switch(P2)} \rightsquigarrow \mathcal{S}_{>950}^{CO_2(A)} \rangle$, which is derived from the smart app rule **R14** and accepted by the hypothesis testing. The threshold 950 is easily extracted via semantic analysis of apps, but it would be difficult, if not impossible, for pure mining based approaches to learn it.

Detecting Case 14. A stealthy command in Case 14 tries to turn on the plug *P2* to start the connected fan, which causes the event $\mathcal{E}_{high}^{power(P2)}$. However, Since the feedback event $\mathcal{E}_{on}^{switch(P2)}$ is intercepted by attackers, the switch of *P2* is still at the state $\mathcal{S}_{off}^{switch(P2)}$. Thus, the physical channel e2s correlation $\mathbb{C}6 = \langle \mathcal{E}_{high}^{power(P2)} \rightsquigarrow \mathcal{S}_{on}^{switch(P2)} \rangle$ is violated.

Detecting Case 20. Command Failures (cyber)/Command Interceptions are detected because of violation of the smart app channel e2e correlation $\mathbb{C}38 = \langle \mathcal{E}_{active}^{motion(M2)} \wedge \mathcal{S}_{home}^{mode} \rightarrow \mathcal{E}_{on}^{switch(P4)} \rangle$: the commands are intercepted or not processed by the cyber part, so there are no feedback events $\mathcal{E}_{on}^{switch(P4)}$. In contrast, HAWatcher (Mining Only) cannot learn this correlation and thus misses all instances of this case.

Detecting Case 21. *L1* accepts the turning-on command and sends the feedback event, but due to a physical-part failure or DoE, the light is not on. While most of the instances of Case 21 can be detected as violation of the correlation $\mathbb{C}24 = \langle \mathcal{E}_{low}^{illuminance(MS1)} \rightsquigarrow \mathcal{S}_{off}^{switch(L1)} \rangle$ (since the illuminance keeps low but the light-switch state is on), 3 instances are missed, because the room has been brightened up by natural light (hence, illuminance has already been high) when the anomaly arises.

For Cases 1, 3, 6, 9, and 10, some instances are missed, which should be attributed to *imperfection of anomaly simulation* (rather than the inability of HAWatcher). For example, seven instances of Case 1 are missed, because the fake motion-active events of MS1 happen to be injected during the time when there are real events of $\mathcal{E}_{active}^{motion(MS1)}$; such missed instances should not impose hazards, as the events are consistent with the fact that the residents are active during the time. Similarly, the 26 missed instances of Case 10 are illuminance readings which have similar values with real readings at the time. For Case 9, two instances are missed because two residents are back home together when one of their presence sensors' events get intercepted. In this situation, smart app **R17** will be triggered without difference by the other presence sensor and no hazard is caused.

Comparison. (1) As shown in Figure 8, HAWatcher achieves the best performance across all the 24 cases. (2) HAWatcher (Apps Only) merely obtains e2e correlations from smart apps, and can only detect anomalies, such as Command Failures (cyber)/Command Interceptions. It gets 16.67% for both the average precision and recall. (3) HAWatcher (Mining Only) has the *second best* performance. On average, its precision is 88.42% and recall 88.62%, showing the effectiveness and importance of our mining approach. However, due to the lack of knowledge of smart apps, it misses many instances of Cases 2, 11, 12, and 20. (4) The ARM-based detector has an average precision 2.03% and recall 7.79%. It fails to detect any anomaly instances for 17 of the 24 cases, as its rules cover very few attributes (Section 6.2). (5) OCSVM performs slightly better with precision 17.15% and recall 45.19%. It fails for Cases 4, 9, 10, and 18, as events related to these cases do not fall inside the same input vector.

False Alarm Rate. We measure the false alarm rate of HAWatcher using the testing event logs (collected during the fourth week). We consider any alarms that are not due to our anomaly injection and cannot be categorized as any of the anomaly types listed in Section 3 as *false alarms*. HAWatcher reports totally 13 anomalies other than those injected by us. Among them, six (6) are due to violations of correlations $\mathbb{C}12$, $\mathbb{C}13$, $\mathbb{C}29$, and $\mathbb{C}15$, because of the large delays of some events from the illuminance sensors; three (3) are due to violations of correlations $\mathbb{C}20$ and $\mathbb{C}21$, because of the large delays of some events from the acceleration sensors. Such anomalies are categorized as true positives due to *Event Losses or Large Delays* (Section 3.1). They should be reported to users, as the large delay may confuse users and even cause undesired automation (e.g., an unlock-door command arrives late after the user has locked the door).

The other four (4) are due to user behavioral deviations: two are due to violation of $\mathbb{C}4$ and $\mathbb{C}5$, because there is one time that the residents stayed outside the door for a while (longer than 60 seconds) before opening the front door; $\mathbb{C}11$ and $\mathbb{C}18$ each cause one false alarm, and the reason is that the residents left the front door open for quite a while and then closed it. While it is arguable whether anomalies due to user behavioral deviations should be categorized as false alarms, we consider them false alarms, as they are not due to attacks or device malfunctions.

In total, HAWatcher reports four (4) false alarms from 9,756

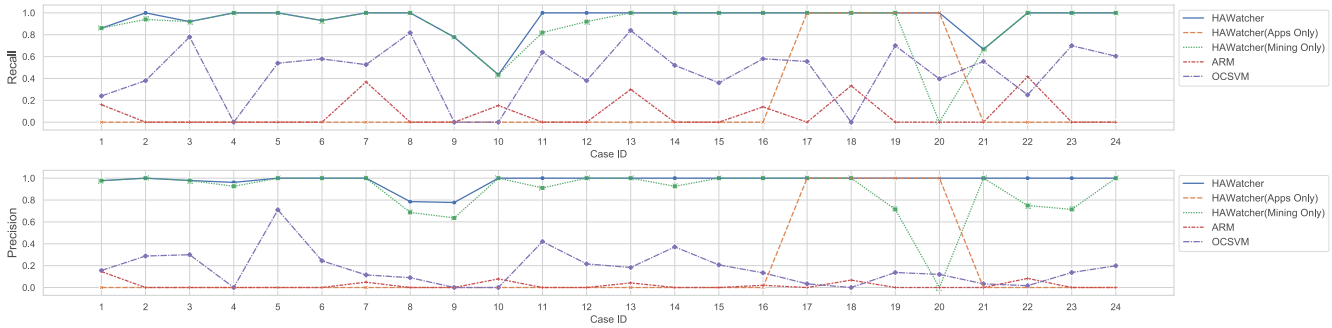

Figure 8: Recall and precision of HAWatcher and four other detectors for comparison purposes.

events collected during a week, which makes 0.57 false alarms per day and a false alarm rate of 0.04%. In comparison, ARM and OCSVM cause 722 and 1,116 false alarms, respectively; that is, 103 and 159 per day and false alarm rates 7.40% and 11.44%, respectively.

6.5 Performance upon Smart App Changes

In an appified home, it is common that users change the smart apps, such as installing new apps and changing the configuration. However, traditional mining based anomaly detection needs a long time to adapt to the changes and, during the adaptation time, may trigger many false alarms. Handling such changes for anomaly detection in appified homes has been challenging. We conduct smart app change experiments to evaluate HAWatcher's performance and compare it with other systems, OCSVM and ARM.

As listed in Table 8, we create five cases of smart app changes, which cover changes of trigger, condition, action, and the whole rule. For each case, we use one day to collect the data, and then apply HAWatcher, OCSVM, and ARM to the collected data. The results show that HAWatcher does not trigger any alarms, while OCSVM triggers many alarms for all the five cases and ARM for the changes of **R8** and **R10**. We manually inspect the alarms and confirm that they are all false alarms caused by app changes.

ARM does not trigger false alarms for the changes of **R3**, **R5**, and **R14** because it does not include any association rules covering the devices, such as *L1* and *L3*, involved in the updated rules. For the OCSVM-based detector, each vector contains four consecutive snapshots of device states. In the case of **R3**, for example, the missing $\mathcal{E}_{on}^{switch(L1)}$ causes unseen vectors and thus triggers false alarms. For HAWatcher, upon app changes, the semantics of the updated apps are extracted and an updated set of correlations obtained. Thus, it is able to handle the changes without triggering false alarms.

7 Related Work

With the emerging development of IoT devices and appified home automation, their security and privacy issues have drawn great attention [28, 29, 34, 50, 57, 61, 73, 74, 78, 79]. Most of them are focused on detecting threats, attacks and malware, rather than IoT malfunctions. For example, HomeGuard [33, 34] presents the first systematic categorization of threats due to interference between different automation apps, dubbed *cross-app interference* (CAI) threats, such as automation conflicts, chained execution, and loop triggering; it is also the first that uses SMT solvers to systematically detect such threats. It conducts symbolic execution to extract automation rules from apps, which is used in this work.

PFirewall [32] is a unique work that notices excessive IoT device data continuously flows to IoT automation platforms. It enforces data minimization, without changing IoT devices or platforms, to protect user privacy from platforms.

IoTSan [61] statically analyzes smart apps to predict whether the resulting automation may violate any safety properties. IoTGuard [29] instruments smart apps. Before an app issues a sensitive command, the action has to pass the policies defined by users. Both rely on pre-defined policies, while HAWatcher does not. Unlike our work, which detects IoT device anomalies, HoMonit [79] is focused on detecting misbehaving smart apps. Given a physical event, Orpheus [31] checks the system call trace due to the event against an automaton to detect attacks; it cannot detect anomalies such as fake events, event interceptions, etc.

Many anomaly detection detectors learn normal behaviors of a smart home from its historical data [26, 35, 51, 54, 60, 69, 75, 76]. For example, SMART [51] trains multiple user activity classifiers based on different subsets of sensor readings, and further trains another classifier that takes the vector of activity-classification results as its input to detect sensor failures. DICE [35] detects anomalies during state transitions by checking the context. Peeves [26] makes use of data from an ensemble of sensors to detect spoofed events.

The main difference of these existing anomaly detectors and our work is that HAWatcher extracts various semantics (device types, device relations, smart apps and their configuration), and infuses the semantics into the mining process. Not only is the detection more accurate, but each detected anomaly can be interpreted as a violation of a correlation, which itself is explainable. Prior to our work, it is unclear how a mining based approach is able to accurately learn complex behaviors in an appified home (e.g., Testbed 1 with 17 apps). HAWatcher provides an effective solution.

Table 8: The number of false alarms caused by smart app changes.

Original Rule	Type	Rule after change	HAWatcher	OCSVM	ARM
R3	Action change	If MS1(active), then L2(on) ~~and L1(on)~~	0	14	0
R5	New rule	If ~~MS2(active)~~ B2(click), then ~~L3(on)~~ L3(toggle)	0	10	0
R8	Condition change	If MS3(inactive) for ~~5~~ 15 minutes, then L4(off)	0	30	67
R10	Condition change	If MS4(inactive) for ~~15~~ 30 minutes, then L5(off)	0	17	75
R14	Trigger change	If A(CO2 > ~~950~~ 1000), then P2(on) for 15 minutes	0	17	0

8 Limitations and Future Work

While the evaluation results are very promising, we consider this work a first step towards semantics-aware anomaly detection in appified smart homes. HAWatcher has some limitations that we plan to address.

User Activity Deviations. Correlations due to the user activity channel are useful for detecting anomalies, but they can cause false alarms when there are user-activity deviations. We already find such cases during our evaluation (see ***False Alarm Rate*** in Section 6.4), although they occur rarely. Some alarms help remind users of unusual situations (e.g., the front door is left open), while others may be annoying. For example, one day a resident wants to read a book in her bedroom and turns on extra lights, which causes illuminance high. If this never or rarely occurs during training, it can cause a false alarm. One potential solution is to ask for users' feedback when raising alarms, and deactivate or re-test correlations that have caused negative feedback. Generally, how to continuously update correlations to adapt to changes of IoT devices and user activities is an important problem.

Long-term Correlations. HAWatcher can only mine correlations whose anterior and posterior events arise within short intervals. Long-interval correlations, such as the relation between turning on AC and temperature events, cannot be mined yet. We can annotate the corresponding cells in the adjacency table with long intervals and use the information during hypothesis testing.

Attackers with More Knowledge. An attacker who knows the correlations may construct attacks that do not violate any correlations in order to evade detection. The bottom line of running HAWatcher is that it imposes extra constraints on attackers. In Testbe 1, each attribute is involved in at least four (4) correlations and has an average of 10.5 correlations (Section 6.2). It is a barrier to attack an device without violating any of the correlations. For example, given the correlation $\langle \mathcal{E}^{lock(frontdoor)}_{unlocked} \rightsquigarrow \mathcal{S}^{presence}_{present} \rangle$ (i.e., the front door unlock event can only arise when the presence sensor is on), if an attacker has compromised the door lock, an alarm will be triggered if the attacker unlocks the door when nobody is home.

Sparsely Deployed IoT Devices. Some IoT devices might be sparsely deployed, and physical-channel correlations among them might be very few. A promising solution is to explore the correlations in the entire home, rather than in separate rooms, which can hopefully derive more correlations among devices. Moreover, it is a trend that IoT devices are deployed with increasing density.

9 Conclusion

In an appified smart home, there exists rich semantic information, such as smart apps, configurations, device types, and installation locations. It is a promising direction to combine such semantic information with mining for anomaly detection. We presented a viable and effective approach in this direction: it exploits semantics on different channels (smart-app, physical, and user-activity) to propose explainable hypothetical correlations, which are tested using event logs and refined by smart apps. We built a prototype *HAWatcher* and evaluated it on four real-world testbeds against various (totally 62) anomaly cases, demonstrating its high accuracy and low false alarm rate. We view this work as a first step, rather than the final solution, in the direction of semantics-aware anomaly detection for appified smart homes.

Acknowledgement

We thank the reviewers for their invaluable suggestions. This work was supported in part by the US National Science Foundation (NSF) under grants CNS-1828363, CNS-1564128, CNS-1824440, CNS-2016589, CNS-1856380 and CNS-2016415.

References

[1] Smartapp execution scheduling. https://docs.smartthings.com/en/latest/ref-docs/smartapp-ref.html#smartapp-run-in.

[2] Lights follows me, 2015. https://github.com/SmartThingsCommunity/SmartThingsPublic/tree/master/smartapps/smartthings/light-follows-me.src.

[3] Door knocker going crazy, 2016. https://community.smartthings.com/t/door-knocker-going-crazy/55570.

[4] Tons of issues with smartthings, 2016. https://www.reddit.com/r/SmartThings/comments/4463eo/anyone_else_having_tons_of_issues_with_smartthings/.

[5] When st glitches become major safety fire hazard, 2016. https://community.smartthings.com/t/when-st-glitches-become-major-safety-fire-hazard/43109.

[6] Are the poltergeists back?, 2017. https://community.smartthings.com/t/october-2017-are-the-poltergeists-back-devices-randomly-turning-on/101402.

[7] Command received but not executed, 2017. https://community.smartthings.com/t/command-received-but-not-executed/112045.

[8] Mobile device presence update delay, 2017. https://community.smartthings.com/t/mobile-device-presence-update-delay/98672.

[9] Motion sensor stuck on motion, 2017. https://community.smartthings.com/t/motion-sensors-stuck-on-motion/46761.

[10] Motion sensors losing connectivity, 2017. https://community.smartthings.com/t/smartthings-motion-multi-sensors-losing-connectivity-on-a-daily-basis/84512.

[11] Tplink smart wi-fi plug fail, 2017. https://www.h3-digital.com/smarthomeblog/2017/5/23/tplink-smart-wi-fi-plug-fail.

[12] Undesired poltergeist lighting effect, 2017. https://community.smartthings.com/t/undesired-poltergeist-lighting-effect/24132.

[13] What's wrong with smartthings now?, 2017. https://community.smartthings.com/t/whats-wrong-with-smartthings-now-poltergeist-events/83889.

[14] Your hotspot is a presence detector. http://ficara.altervista.org/?p=3744&doing_wp_cron=1591921359.5108020305633544921875, 2017.

[15] It's too cold, 2018. https://github.com/SmartThingsCommunity/SmartThingsPublic/tree/master/smartapps/smartthings/its-too-cold.src.

[16] Light up the night, 2018. https://github.com/infinitywings/SmartThingsPublic/blob/master/smartapps/smartthings/light-up-the-night.src/light-up-the-night.groovy.

[17] Motion detection false positive, 2018. https://community.smartthings.com/t/motion-detection-false-positive/119816.

[18] Smart plug clicks but no power, 2018. https://community.smartthings.com/t/smart-plug-clicks-but-no-power/115252.

[19] Smartthings capabilities, 2018. https://smartthings.developer.samsung.com/docs/api-ref/capabilities.html.

[20] Known mobile presence issues and faq, 2019. https://support.smartthings.com/hc/en-us/articles/204744424-Known-mobile-presence-issues-and-FAQ.

[21] Smartthings, 2019. https://www.smartthings.com.

[22] Pymining - a collection of data mining algorithms in python, 2020. https://github.com/bartdag/pymining.

[23] Troubleshooting: Smartthings multipurpose sensor is stuck on "open" or "closed", 2020. https://support.smartthings.com/hc/en-us/articles/200955940-Troubleshooting-SmartThings-Multipurpose-Sensor-is-stuck-on-open-or-closed-.

[24] Rakesh Agrawal, Ramakrishnan Srikant, et al. Fast algorithms for mining association rules. In *Proceedings of 20th International Conference of Very Large Data Bases (VLDB)*, volume 1215, pages 487–499, 1994.

[25] Omar Alrawi, Chaz Lever, Manos Antonakakis, and Fabian Monrose. Sok: Security evaluation of home-based iot deployments. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2019.

[26] Simon Birnbach, Simon Eberz, and Ivan Martinovic. Peeves: Physical event verification in smart homes. In *Proceedings of the ACM Conference on Computer & Communications Security (CCS)*, pages 1455–1467, 2019.

[27] Kate Calder. Statistical inference. *New York: Holt*, 1953.

[28] Z Berkay Celik, Patrick McDaniel, and Gang Tan. Soteria: Automated iot safety and security analysis. In *2018 USENIX Annual Technical Conference (USENIX ATC)*, pages 147–158, 2018.

[29] Z Berkay Celik, Gang Tan, and Patrick D McDaniel. Iotguard: Dynamic enforcement of security and safety policy in commodity iot. In *Network and Distributed System Security Symposium (NDSS)*, 2019.

[30] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):15, 2009.

[31] Long Cheng, Ke Tian, and Danfeng Daphne Yao. Orpheus: Enforcing cyber-physical execution semantics to defend against data-oriented attacks. In *Proceedings of the 33rd Annual Computer Security Applications Conference (ACSAC)*, pages 315–326, 2017.

[32] Haotian Chi, Qiang Zeng, Xiaojiang Du, and Lannan Luo. PFirewall: Semantics-aware customizable data flow control for home automation systems. *arXiv preprint arXiv:1910.07987*, 2019.

[33] Haotian Chi, Qiang Zeng, Xiaojiang Du, and Jiaping Yu. Cross-app interference threats in smart homes: Categorization, detection and handling. *arXiv*, pages arXiv–1808, 2018.

[34] Haotian Chi, Qiang Zeng, Xiaojiang Du, and Jiaping Yu. Cross-app interference threats in smart homes: Categorization, detection and handling. In *50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 411–423, 2020.

[35] Jiwon Choi, Hayoung Jeoung, Jihun Kim, Youngjoo Ko, Wonup Jung, Hanjun Kim, and Jong Kim. Detecting and identifying faulty iot devices in smart home with context extraction. In *48th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2018.

[36] Diane J Cook, Aaron S Crandall, Brian L Thomas, and Narayanan C Krishnan. Casas: A smart home in a box. *Computer*, 46(7):62–69, 2013.

[37] Diane J Cook, Michael Youngblood, Edwin O Heierman, Karthik Gopalratnam, Sira Rao, Andrey Litvin, and Farhan Khawaja. Mavhome: An agent-based smart home. In *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pages 521–524, 2003.

[38] Borden Dent. Cartography–thematic map design. 1999. pages 147–149.

[39] Wenbo Ding and Hongxin Hu. On the safety of iot device physical interaction control. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer & Communications Security (CCS)*, pages 832–846, 2018.

[40] Nancy ElHady and Julien Provost. A systematic survey on sensor failure detection and fault-tolerance in ambient assisted living. *Sensors*, 18(7):1991, 2018.

[41] Earlence Fernandes, Jaeyeon Jung, and Atul Prakash. Security analysis of emerging smart home applications. In *IEEE Symposium on Security and Privacy (S&P)*, pages 636–654, 2016.

[42] Ronald Aylmer Fisher. Statistical methods for research workers. In *Breakthroughs in statistics*, pages 66–70. Springer, 1992.

[43] Milan Fránik and Miloš Čermák. Serious flaws found in multiple smart home hubs: Is your device among them?, 2020. https://www.welivesecurity.com/2020/04/22/serious-flaws-smart-home-hubs-is-your-device-among-them/.

[44] Chenglong Fu, Qiang Zeng, and Xiaojiang Du. Hawatcher: Semantics-aware anomaly detection for appified smart homes (technical report), 2020. https://github.com/infinitywings/HAWatcher.git.

[45] Jun Han, Albert Jin Chung, Manal Kumar Sinha, Madhumitha Harishankar, Shijia Pan, Hae Young Noh, Pei Zhang, and Patrick Tague. 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)*, pages 836–852, 2018.

[46] Timothy W Hnat, Vijay Srinivasan, Jiakang Lu, Tamim I Sookoor, Raymond Dawson, John Stankovic, and Kamin Whitehouse. The hitchhiker's guide to successful residential sensing deployments. In *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, pages 232–245, 2011.

[47] Apple Homekit. Homekit-apple developer, 2019. https://www.apple.com/ios/home/.

[48] Jun Inoue, Yoriyuki Yamagata, Yuqi Chen, Christopher M Poskitt, and Jun Sun. Anomaly detection for a water treatment system using unsupervised machine learning. In *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, pages 1058–1065, 2017.

[49] George F Jenks. The data model concept in statistical mapping. *International yearbook of cartography*, 7:186–190, 1967.

[50] Yunhan Jack Jia, Qi Alfred Chen, Shiqi Wang, Amir Rahmati, Earlence Fernandes, Z Morley Mao, Atul Prakash, and Shanghai JiaoTong Unviersity. Contexiot: Towards providing contextual integrity to appified iot platforms. In *Proceedings of The Network and Distributed System Security Symposium (NDSS)*, 2017.

[51] Krasimira Kapitanova, Enamul Hoque, John A Stankovic, Kamin Whitehouse, and Sang H Son. Being smart about failures: assessing repairs in smart homes. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing (UbiComp)*, pages 51–60, 2012.

[52] Stylianos P Kavalaris and Emmanouil Serrelis. Security issues of contemporary multimedia implementations: The case of sonos and sonosnet. In *The International Conference in Information Security and Digital Forensics (ISDF)*, pages 63–74, 2014.

[53] Shehroz S Khan and Michael G Madden. One-class classification: taxonomy of study and review of techniques. *The Knowledge Engineering Review*, 29(3):345–374, 2014.

[54] Palanivel A Kodeswaran, Ravi Kokku, Sayandeep Sen, and Mudhakar Srivatsa. Idea: A system for efficient failure management in smart iot environments. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys)*, pages 43–56, 2016.

[55] K Kreuzer. Openhab-empowering the smart home, 2013.

[56] Wenke Lee, Salvatore J Stolfo, and Kui W Mok. A data mining framework for building intrusion detection models. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, pages 120–132, 1999.

[57] Xiaopeng Li, Qiang Zeng, Lannan Luo, and Tongbo Luo. T2Pair: Secure and Usable Pairing for Heterogeneous IoT Devices. In *Proceedings of the ACM Conference on Computer & Communications Security (CCS)*, 2020.

[58] Haoyu Liu, Tom Spink, and Paul Patras. Uncovering security vulnerabilities in the belkin wemo home automation ecosystem. In *2019 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pages 894–899, 2019.

[59] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems (NeurIPS)*, pages 3111–3119, 2013.

[60] Sirajum Munir and John A Stankovic. Failuresense: Detecting sensor failure using electrical appliances in the home. In *11th International Conference on Mobile Ad Hoc and Sensor Systems (MobiHoc)*, pages 73–81, 2014.

[61] Dang Tu Nguyen, Chengyu Song, Zhiyun Qian, Srikanth V Krishnamurthy, Edward JM Colbert, and Patrick McDaniel. Iotsan: fortifying the safety of iot systems. In *Proceedings of the 14th International Conference on emerging Networking Experiments and Technologies (CoNEXT)*, pages 191–203, 2018.

[62] Sukhvir Notra, Muhammad Siddiqi, Hassan Habibi Gharakheili, Vijay Sivaraman, and Roksana Boreli. An experimental study of security and privacy risks with emerging household appliances. In *IEEE conference on communications and network security (CNS)*, 2014.

[63] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[64] Friedrich Pukelsheim. The three sigma rule. *The American Statistician*, 48(2):88–91, 1994.

[65] Eyal Ronen, Adi Shamir, Achi-Or Weingarten, and Colin O'Flynn. Iot goes nuclear: Creating a zigbee chain reaction. In *2017 IEEE Symposium on Security and Privacy (S&P)*, pages 195–212, 2017.

[66] Lee Russell. Wireless security monitoring versus a cellular jammer. 2014.

[67] Bernhard Schölkopf, John C Platt, John Shawe-Taylor, Alex J Smola, and Robert C Williamson. Estimating the support of a high-dimensional distribution. *Neural computation*, 13(7):1443–1471, 2001.

[68] Amit Kumar Sikder, Hidayet Aksu, and A Selcuk Uluagac. 6thsense: A context-aware sensor-based attack detector for smart devices. In *26th USENIX Security Symposium (USENIX Security)*, pages 397–414, 2017.

[69] Amit Kumar Sikder, Leonardo Babun, Hidayet Aksu, and A Selcuk Uluagac. Aegis: a context-aware security framework for smart home systems. In *Proceedings of the 35th Annual Computer Security Applications Conference (ACSAC)*, pages 28–41, 2019.

[70] Vijay Sivaraman, Dominic Chan, Dylan Earl, and Roksana Boreli. Smart-phones attacking smart-homes. In *Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks (WiSec)*, pages 195–200, 2016.

[71] Yuan Tian, Nan Zhang, Yueh-Hsun Lin, XiaoFeng Wang, Blase Ur, Xianzheng Guo, and Patrick Tague. Smartauth: User-centered authorization for the internet of things. In *26th USENIX Security Symposium (USENIX Security)*, pages 361–378, 2017.

[72] Rob van der Meulen and Janessa Rivera. Gartner says a typical family home could contain more than 500 smart devices by 2022. Technical report, 2014. http://www.gartner.com/newsroom/id/2839717.

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

[74] Rixin Xu, Qiang Zeng, Liehuang Zhu, Haotian Chi, Xiaojiang Du, and Mohsen Guizani. Privacy leakage in smart homes and its mitigation: Ifttt as a case study. *IEEE Access*, 7:63457–63471, 2019.

[75] Moosa Yahyazadeh, Proyash Podder, Endadul Hoque, and Omar Chowdhury. Expat: Expectation-based policy analysis and enforcement for appified smart-home platforms. In *Proceedings of the 24th ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 61–72, 2019.

[76] Juan Ye, Graeme Stevenson, and Simon Dobson. Fault detection for binary sensors in smart home environments. In *Pervasive Computing and Communications (PerCom)*, pages 20–28, 2015.

[77] Juan Ye, Graeme Stevenson, and Simon Dobson. Detecting abnormal events on binary sensors in smart home environments. In *Pervasive and Mobile Computing*, pages 32–49, 2016.

[78] Qiang Zeng, Jianhai Su, Chenglong Fu, Golam Kayas, Lannan Luo, Xiaojiang Du, Chiu C Tan, and Jie Wu. A multiversion programming inspired approach to detecting audio adversarial examples. In *49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 39–51, 2019.

[79] Wei Zhang, Yan Meng, Yugeng Liu, Xiaokuan Zhang, Yinqian Zhang, and Haojin Zhu. Homonit: Monitoring smart home apps from encrypted traffic. In *ACM SIGSAC Conference on Computer & Communications Security (CCS)*, pages 1074–1088, 2018.

[80] Wei Zhou, Yan Jia, Yao Yao, Lipeng Zhu, Le Guan, Yuhang Mao, Peng Liu, and Yuqing Zhang. Discovering and understanding the security hazards in the interactions between iot devices, mobile apps, and clouds on smart home platforms. In *28th USENIX Security Symposium (USENIX Security)*, pages 1133–1150, 2019.

A Experimental Results of Testbeds 2 to 4

Table 9: Smart apps deployed on Tesbeds 2 $\sim$ 4. **R2.1**, for example, means the first smart app rule on Testbed 2.

Index	Smart app rules
R2.1	If MS2(active), then P1(on) and L1(on)
R2.2	If MS2(inactive) for 30 minutes, then P1(off), L1(off), L2(off), L3(off)
R2.3	If MS3(active), then L4(on)
R2.4	If MS3(inactive) for 10 minutes, then L4(off)
R2.5	If W(wet) or MS3(humidity$\geq$55), then V(on)
R2.6	If V(on) for 15 minutes, then V(off)
R2.7	If PS1(present) or PS2(present), then turn on L1, L2, L5, P1
R2.8	If PS1(away) and PS2(away), then turn off L1, L2, L3, L4, L5, V, P1
R2.9	If B(pressed), toggle L5
R2.10	If B(held), then turn off all L and P
R2.11	If B(double pressed), turn on L1 and L5 and P1
R3.1	If MS1(active) and Mode(home), then L1(on)
R3.2	If MS1(inactive) for 60 minutes, then L1(off)
R3.3	If B(pressed), toggle L1
R3.4	If B(held), then L1(off) and Mode(night)
R3.5	If B(double pressed), then L1(on) Mode(home)
R3.6	If PS(away), then L1(off), P1(off), and Mode(away)
R3.7	If PS(present), then L1(on), P1(on), and Mode(home)
R4.1	If PS(away), then P1(off) and P2(off)
R4.2	If PS(present) then P1(on), P2(on)
R4.3	If B(pressed), toggle P1
R4.4	If B(held), toggle P2

A.1 Deployment

We list the smart apps deployed on Testbeds 2, 3, and 4 in Table 9. On Testbed 3, the *mode* is used as a condition to control the behavior of the light, while Testbeds 2 and 4 do not use the mode. Since Testbed 2 has two residents, lights and plugs are only turned off when both residents are away (**R2.8**). Testbeds 3 and 4 have one resident each, and all lights and plugs are turned off when the resident leaves home.

A.2 Training and Testing Results

On Testbed 2, we extract 32 e2e correlation from smart apps and pass 98 correlations from 2064 hypothetical correlations. In total, we get 109 correlations after refining. The difference of correlations regarding contact sensors, as observed on Testbed 1, is also observed on Testbed 2: *C1* on the front door always gets closed right after the acceleration is detected, while *C2* and *C3* are usually left open for a long time. The inaccurate correlation $\langle \mathcal{E}_{away}^{presence(PS2)} \rightarrow \mathcal{E}_{off}^{switch(L1)} \rangle$ is accepted by the hypothesis testing. If not refined by the smart app rule **R2.8**, it causes 4 false alarms for HAWatcher (Mining Only) on case **2.3** and **2.6** when only the resident taking *PS2* leaves home. As detailed in our technical report [44], HAWatcher achieves an average detection precision of 94.85% and recall of 96.86%. In terms of the false alarm test, HAWatcher raises 13 false alarms among 6721 events collected within one week's testing period, which causes a false alarm rate (FAR) of 0.19% and 1.86 false alarms per day. Among the 13 false alarms, four (4) are raised by the correlations $\langle \mathcal{E}_{active}^{acceleration(C1)} \rightarrow \mathcal{E}_{active}^{motion(MS1)} \rangle$ and $\langle \mathcal{E}_{active}^{acceleration(C2)} \rightarrow \mathcal{E}_{active}^{motion(MS2)} \rangle$ because of strong vibrations in the neighborhood that trigger events of the acceleration sensor *C1* and *C2*. Three (3) are raised by $\langle \mathcal{E}_{low}^{illuminance(L4)} \rightsquigarrow \mathcal{S}_{inactive}^{motion(MS3)} \rangle$ because there are three times that a resident remains active in the study room after the light is turned off. Four (4) are caused by $\langle \mathcal{E}_{closed}^{contact(C3)} \rightsquigarrow \mathcal{S}_{active}^{motion(MS3)} \rangle$ because residents close the door from outside. In contrast, the OCSVM-based detector has an average precision of 11.11% and recall of 35.41% with 968 false alarms raised. The ARM-based detector has an average precision of 3.76% and a recall of 9.96%, and raises 370 false alarms.

On Testbed 3, HAWatcher accepts 50 correlations from 527 hypotheses, and 15 e2e correlations from smart apps. After refining, there are 55 correlations left. HAWatcher achieves an average detection precision of 92.74% and a recall of 93.36%. Among the testing period, ten (10) false alarms are raised by HAWatcher among 2411 events, which leads to 1.42 false alarms per day on average and a FAR of 0.42%. In contrast, the OCSVM-based detector has an average precision of 31.01% and a recall of 42.33%, and raises 379 false alarms. The ARM-based detector has an average precision of 9.89% and an average recall of 14.10%, and raises 152 false alarms.

On Testbed 4, only 26 correlations are acquired because of the low density of IoT devices and smart apps. HAWatcher gets an average detection precision of 96.62% and a recall of 90.17%. Five (5) false alarms are raised on this testbed among 1674 events, that is, 0.71 false alarms per day and a FAR of 0.30%. In contrast, the OCSVM-based detector has an average precision of 28.80% and a recall of 42.37%, and raises 168 false alarms. The ARM-based detector has an average precision of 3.60% and a recall of 7.38%, and raises 108 false alarms.