BIG DATA MINING AND ANALYTICS
ISSN 2096-0654 08/10 pp218–233
Volume 6, Number 2, June 2023
DOI: 10.26599/BDMA.2022.9020034

EScope: Effective Event Validation for IoT Systems Based on State Correlation

Jian Mao*, Xiaohe Xu, Qixiao Lin, Liran Ma, and Jianwei Liu

Abstract: Typical Internet of Things (IoT) systems are event-driven platforms, in which smart sensing devices sense or subscribe to events (device state changes), and react according to the preconfigured trigger-action logic, as known as, automation rules. "Events" are essential elements to perform automatic control in an IoT system. However, events are not always trustworthy. Sensing fake event notifications injected by attackers (called event spoofing attack) can trigger sensitive actions through automation rules without involving authorized users. Existing solutions verify events via "event fingerprints" extracted by surrounding sensors. However, if a system has homogeneous sensors that have strong correlations among them, traditional threshold-based methods may cause information redundancy and noise amplification, consequently, decreasing the checking accuracy. Aiming at this, in this paper, we propose "EScope", an effective event validation approach to check the authenticity of system events based on device state correlation. EScope selects informative and representative sensors using an Neural-Network-based (NN-based) sensor selection component and extracts a verification sensor set for event validation. We evaluate our approach using an existing dataset provided by Peeves. The experiment results demonstrate that EScope achieves an average 67% sensor amount reduction on 22 events compared with the existing work, and increases the event spoofing detection accuracy.

Key words: Internet of Things (IoT); event spoofing; event fingerprint; correlation analysis

1 Introduction

In a nutshell, Internet of Things (IoT) is the network consisting of "things" (physical objects that are embedded with sensors), software, and other techniques for the purpose of connecting, controlling, and exchanging data with other devices or systems over Internet[1]. In typical IoT systems (e.g., smart home, industrial IoT system, etc.), devices usually include sensors that sense the physical environment[2, 3] and activators that interact with the environment. To enable automatic control in IoT systems, Trigger-Action Programming (TAP) logic is deployed in the IoT control system (e.g., a cloud back-end or a local server). Typical TAP logic-enabled IoT platforms are event-driven, such as Samsung SmartThings[4] and open source home automation platform Home Assistant[5]. In these event-driven platforms[6], an event-bus is used as the information hub, and events are published by devices to inform the state changes to the IoT control system. When a device has a state change, it sends an event notification to the event bus. According to automation rules, devices subscribe trigger events and perform actions correspondingly.

• Jian Mao, Xiaohe Xu, Qixiao Lin, and Jianwei Liu are with the School of Cyber Science and Technology, Beihang University, Beijing 100191, China. Jian Mao and Qixiao Lin are also with the Beihang Hangzhou Innovation Institute Yuhang, Hangzhou 310023, China. E-mail: {maojian, 16711110, linqx529, liujianwei}@buaa.edu.cn.
• Liran Ma is with the Department of Computer Science, Texas Christian University, Fort Worth, TX 76129, USA. E-mail: l.ma@tcu.edu.
* To whom correspondence should be addressed.
Manuscript received: 2022-07-28; revised: 2022-09-23; accepted: 2022-09-26

As events are received by the IoT control systems for subsequent actions, the authenticity of event notifications is essential to IoT system security. Recently, several event-related security threats have been reported[7], such as event losses[8], event interceptions[9], and event spoofing[10]. Typically, event spoofing is that attackers inject and send forged event notifications via malicious apps[10,11] or compromised devices[12]. Sensing mistakes caused by devices or fake event notifications injected by attackers (called event spoofing attack) can trigger sensitive actions through user pre-defined automation rules without user authorization[13].

In IoT systems, the event notifications are accepted by the control system without validation. This allows attackers successfully launch event spoofing attacks. To validate an event notification and detect event spoofing attacks, event fingerprint-based solutions are proposed. Existing approaches explore the inner-relationship between events and data from different sources, including encrypted network traffic data[14–17] and heterogeneous sensor data[13,18–20], and validate events using classification[21] or clustering methods.

Intuitively, a physical event will inevitably affect surrounding sensor readings. Such influences on assemble devices can be extracted and formulated as fingerprints for event validation. Existing fingerprint-based event validation methods use a fixed time window to select sensor data and extract features from the data sequence. Informative devices are selected as "verification sensor set" according to correlation metrics (such as mutual information) and a preset threshold for validation. However, due to the neglect of inter-sensor correlations, if a system has homogeneous sensors that have strong correlations among them, such threshold-based methods may select similar sensors repeatedly. As a result, the verification sensor set consists of large amount of high-informative but correlated sensors, rather than a small amount of independent and representative sensors, which will bring information redundancy and amplify noisy features. Consequently, this decreases the checking accuracy. How to determine a "verification sensor set" that consists of informative and independent sensors is the key problem in fingerprint-based event validation approaches.

In this paper, we propose "EScope", an effective event validation approach to check the authenticity of system events based on device state correlation. EScope filters informative and efficient sensors using an Neural-Network-based (NN-based) sensor selection component and extracts a verification sensor set for event validation. We implement and evaluate EScope based on a public dataset provided by Peeves[13]. The experiment results show that EScope achieves an average 67% sensor amount reduction on 22 events compared to Peeves[13]. Meanwhile, EScope increases the F1-score and Area Under Curve (AUC) by more than 0.1. We simulate typical event spoofing attacks and demonstrate that EScope can more effectively detect event spoofing attacks.

Our contributions are as follows:

- We identify the key problem of fingerprint-based event verification based on systematic analysis, and propose EScope, an effective event validation approach to check the authenticity of system events based on device state correlation.
- We develop an NN-based sensor selection method that effectively selects informative and representative sensors for event validation.
- We evaluated our approach using an existing dataset provided by Peeves. The experiment results demonstrate that EScope performs more accurately than existing solutions with fewer verification sensors.

The rest of this paper is organized as follows. Section 2 discusses related works. Section 3 discusses the background, preliminaries, and the threat model. Section 4 presents the design of our effective event validation approach, Escope. Section 5 describes the implementation details of Escope. In Section 6, we evaluate Escope and analyze the experiment results. Section 7 concludes the paper.

2 Related Work

(1) Security of IoT devices. IoT devices are usually small in size, power consumption, and computing ability, thus strong cryptography-based authentication or encryption algorithms are hard to implement, which brings unreliability and insecurity. There are extensive researches focusing on the IoTs security. By collecting information on user forums and recent researches, Fu et al.[7] and Can and Zheng[22] concluded typical anomalies of IoT resulting from device malfunctions and adversary attacks. Numerous sensors are the most vulnerable part of the IoT system. Debes et al.[23] proved that sensor data can be used to monitor user Activities of Daily Life (ADL), which could lead to privacy leaks. They also suggested that cameras and microphones are the most influential sensors of user privacy. Islam et al.[24] and Zheng et al.[25] introduced security threats to the wireless

sensor networks, including eavesdropping, denial of service, and node compromise. Sivaraman et al.[26] managed to developed an iOS app to automatically find vulnerable IoT devices in victims' local networks.

(2) Security of IoT events. Fu et al.[7] introduced event-related security threats in IoT systems, including event losses, event interceptions, faulty events, and fake events. Fernandes et al.[10] introduced event spoofing in Samsung SmartThings. They pointed out that when the system do not enforce access control around event notifications or verify the integrity of the origin of the events, an unprivileged entity can spoof device events to escalate its privileges. Gu et al.[27] proposed IoTGase, which uses wireless context to consider security problems in IoT. They simulated event spoofing in their evaluation by inserting malicious code into the apps, proving the effectiveness of their method. Wang et al.[28] proposed IoT-Praetor, a Device Usage Description (DUD) model to detect undesired behavior for IoT devices, which is also effective for event spoofing according to their evaluation.

(3) Traffic data based event fingerprints. Acar et al.[14] proposed a method to steal users' privacy via encrypted network traffic data of different communication protocols in smart homes. The method first recognizes devices of each traffic package using state-of-the-art methods, then recognizes device events based on a machine learning algorithm to further infer users' activities. Zhang et al.[16] proposed a third-party monitoring system called Homonit to detect illegal behavior of SmartApps using encrypted network traffic data, including verifying events sent from these apps. They hypothesized that there is a one-to-one mapping between events and traffic package groups, then further proposed a mathematical model to depict the event fingerprint model, which can be used to calculate features and do matching between events to be verified and a flag value. Trimananda et al.[17] proposed PingPong to extract packet-level signatures from encrypted traffic. They suggested that a unique sequence of packet lengths can usually describe certain simple events and be used as an event fingerprint.

(4) Sensor data based event fingerprints. Yasaei et al.[20] proposed IoT-CAD, which utilizes sensors in IoT systems to capture physical environments to identify anomalies. Sensors are required for continuously monitoring, and snapshot vectors in small time windows are extracted for fingerprints. Laput et al.[18] proposed synthetic sensors to detect events and user's activities from sensor data. They choose different features for high sampling rate sensors and low sampling rate sensors. In the aspects of machine learning algorithms, classification and clustering can both be used to verify the events. Birnbach et al.[13] proposed Peeves that selects time windows and features to build a machine learning dataset, which is for further classification. This is a close related work of our approach.

We summarize and compare the existing fingerprinting approaches in Table 1. Overall, traffic-based fingerprinting methods deploy a centralized gateway that captures packet headers from traffic packets and extracts information such as timestamp, length, address, and protocol as fingerprint features. Such methods aim at behavior inference and abnormal behavior detection. Sensor-data-based fingerprinting methods capture the raw surrounding sensor data sequences and their frequency domain to extract features, set thresholds on indicators such as mutual information to select sensors, and create fingerprinting for validation. However, neglecting the strong correlation among homogeneous sensors, threshold-based sensor selection may cause information redundancy and noise amplification. To solve this problem, EScope proposes an NN-based sensor selecting method that fully considers the contribution to event validation of each sensor. In this way, EScope avoids selecting homogeneous sensors

Table 1 Analysis of existing approaches on event fingerprints.

Fingerprint source	Approach	Packet feature				Sensor featue		Learning algorithm
		TS	L	AD	P	TD	FD	
Network traffic data	Peek-a-boo[14]	✓	✓	×	×	–	–	KNN
	Homonit[16]	✓	✓	✓	×	–	–	DFA
	PingPong[17]	✓	×	×	✓	–	–	DBSCAN
Sensor data	Peeves[13]	–	–	–	-	✓	×	SVM
	Synthetic sensors[18]	–	–	–	-	✓	✓	SVM/EM
	IoT-CAD[20]	–	–	–	–	✓	×	RNN
	EScope (Our approach)	–	–	–	–	✓	×	SVM

Note: TS denotes timestamp; L denotes packet length; AD denotes address; P denotes protocol; TD denotes time domain; and FD denotes frequency domain.

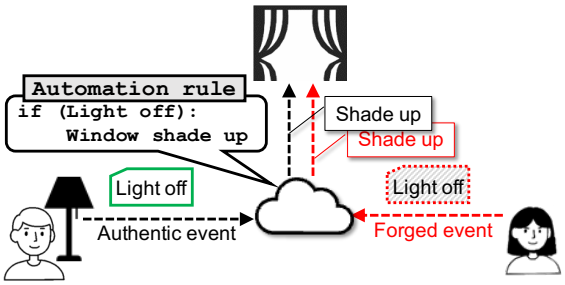
repeatedly, reduces information redundancy, and improves accuracy.

3 Background and Threat Model

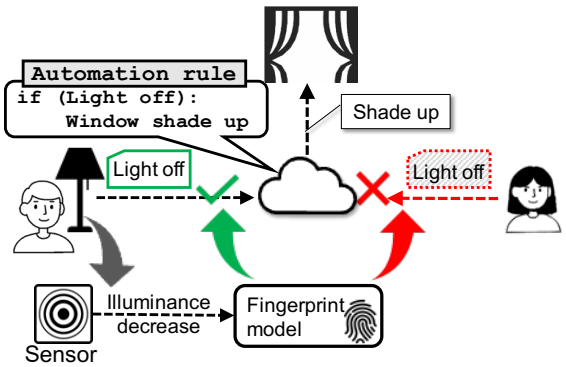
3.1 Background

(1) Event driven mechanisms in IoT systems. Most IoT platforms support both direct control commands and automation rules in the "trigger-condition-action" pattern. For example, a light can be turned on by directly clicking the button in the User Interface (UI), or triggered by automation rules, e.g., "**if** the door is opened **then** turn on the light". To enable a heterogeneous architecture that supports different communication protocols, packet structures, and device working modalities, IoT systems use "events" as essential control elements. Events in an IoT system are denoted as device state changes. For example, when a door is opened, the event "Door opened" is sent to the control system. Such event triggers the automation rule, "**if** the door is opened **then** turn on the light", consequently, the light is turned on and a "Light on" event is sent to the IoT control system.

(2) Event spoofing attacks. A typical event spoofing attack is shown in Fig. 1a. In an IoT system, the attacker sends a forged event notification "Light off" to the control system, triggering the automation rule to make the window shade to go up. The lack of an authentication mechanism to validate the authenticity of an event notification facilitates event spoofing attacks. Intuitively, the authenticity of an event notification can be validated according to the influences on assemble devices caused by the event. For example, in Fig. 1b, a "Light off" event affects the state of the illuminance sensor. Such influences can be extracted and formulated as fingerprints for event validation. If the system receives an event notification without a valid/proper event fingerprint from surrounding sensors (in this example, the illuminance sensor), the notification will be rejected. Accordingly, only valid events (e.g., caused by user operation in Fig. 1b can be processed and trigger automation rules.



(a) Event spoofing attack



(b) Event verification by using sensor fingerprints

Fig. 1 Examples of event spoofing attack and its defense solution.

(3) Sensor state based event fingerprint. Physical events in the real world will cause IoT sensors to fluctuate according to a certain pattern. Further, this impact only happens in a short time window around the event timestamp. As shown in Fig. 2, opening and closing the door will cause the accelerometer to fluctuate, but the acceleration only occurs after the event "Door opened" and before the event "Door closed". The time range indicated by the arrow below represents the time window of the events, and the small flat of the curve in the middle represents the gap between "Door opened" and "Door closed", that is, the door is not subject to acceleration. Therefore, a time window based sensor data feature extraction method can be used to depict the correlation between sensor data and physical events, in which sensor data are used to build event fingerprints. Learning-based methods are usually deployed to extract fingerprint models.

3.2 Threat model

We consider a scenario that the IoT system receives a forged event notification, i.e., the state of the event corresponding device does not actually change. This can further trigger certain IoT automation rules. Event spoofing is a typical measure to inject forged events into the victim IoT system. In this case, the attacker needs to have some preliminaries about the victim, such as

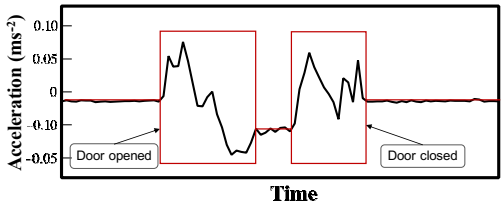


Fig. 2 Sensor data fluctuation caused by events.

the corresponding automation rules of its attack goal, so that the attacker can choose which event to forge. For example, as illustrated in Fig. 1a, if the attacker knows about the rule "**if** the door is opened **then** turn on the light", he/she will exploit a forged "Light off" event to change the window shade state indirectly.

In an event spoofing attack, the attacker is not able to make any device state to change in the physical space, but he/she can deceive the system by sending forged event notifications through cyber-attacks, such as malicious apps, traffic packet forgery, compromised weak sensors, etc. Sivaraman et al.[26] showed that malicious smartphone apps can exploit vulnerable IoT devices through the Internet, which can send forged events to the control system. Some typical local-server-based IoT systems, such as Home Assistant[5], have weak security mechanisms on network traffic, i.e., they usually use unencrypted Hyper Text Transfer Protocol (HTTP) for inter-device communications, which gives opportunities for traffic packet forgery[29]. Furthermore, the attacker can choose to compromise a weak sensor to spoof events to trigger a secure sensitive action[13].

The goal of this paper is to validate event authenticity and detect anomaly event notifications caused by the aforementioned event spoofing attack or device errors.

4 System Design

In this section, we present the system design of our approach, EScope. The goal of our approach is to effectively validate the target event with the assistance of closely related surrounding sensors (i.e., verification sensors). As shown in Fig. 3, EScope consists of five modules: (1) data labelling; (2) feature construction; (3) sensor selection; (4) fingerprint generating; and (5) training and checking. Data labelling module labels the features according to the event logs. Meanwhile, feature construction module takes event timestamps and raw sensor data as input, uses grid search to select optimal sensor windows, and computes features. Sensor selection module trains a neural network to select "representative" features, and selects sensors with corresponding features as the verification sensor set. Fingerprint generating module generates event fingerprints from the selected sensors. Training and checking module trains classifiers and outputs the checking results.

In EScope, we consider each event individually. In this section we denote the target event as E. Assume there are k sensors in the IoT system, we denote their time sequences of sensor s_i as $S_i(t), i \in \{1, 2, \ldots, k\}$. Suppose an event E occurs n times, the timestamp of the j-th event occurrence is denoted as t_j, $j \in \{1, 2, \ldots, n\}$. We summarize the definition of all parameters applied in our approach in Table 2.

4.1 Data labelling

Data labelling module takes event logs as inputs and outputs event tuples. In our approach, an event tuple consists of three elements, an event name E, a timestamp t_j, and a label $l_j \in \{0, 1\}$. Such event tuple means the event status at the moment t. We label the event timestamp as "1" (positive) to represent the event did occur. We label the timestamp between event occurrences at a certain interval as "0" (negative) to represent the event did not occur. We generate event tuples for every n records in the event log, and the corresponding event tuple sequence is denoted as $(E, t_j, l_j), j \in \{1, 2, \ldots, n\}$.

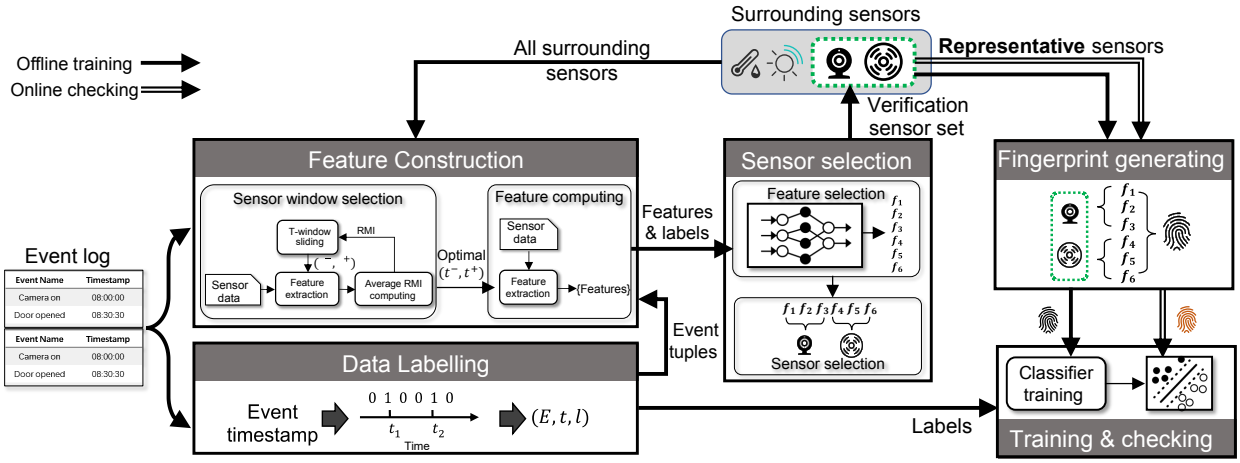


Fig. 3 Overall architecture of EScope.

Table 2 List of the major variables.

Variable	Definition
E	Event to be verified
k	Total amount of sensors
n	Total amount of event occurrences
$S_i(t)$	Data sequence of the i-th sensor s_i
t_j	Timestamp of the j-th event occurrence
l_j	Label of the j-th event occurrence
l	Label sequence composed of l_j
$(t^-_{E,S_i}, t^+_{E,S_i})$	Time window for event E and sensor S_i
$\vec{F_{S_i}}(t_j)$	Feature vector of sensor S_i at t_j
$f_{S_i,m}(t_j)$	The m-th feature value of sensor S_i at t_j
$f_{S_i,m}$	The m-th feature value sequence of sensor S_i
$\vec{F(t_j)}$	Feature vector including all sensors at t_j
$\vec{x}$	Neural network input vector
y	Neural network output value
$\vec{x_0}$	Output vector of the one-to-one layer
$\vec{\omega}$	Weight vector of the one-to-one layer
$\vec{\omega_1}, \vec{\omega_2}, \vec{\omega_3}$	Neural Network weight vectors
$\vec{b_1}, \vec{b_2}, \vec{b_3}$	Neural network bias vectors
J	Neural network loss
η_0, η_1	Neural network learning rates
λ	Neural network L1-regularization penalty

4.2 Feature construction

The purpose of this module is to obtain features and their labels. This module takes event logs and raw sensor data sequences as inputs and outputs features of all sensors and the corresponding labels. This module consists of two sub-components, sensor window selection, and feature computing.

(1) Sensor window selection. This sub-component takes event tuples and raw sensor data as inputs and outputs optimal sensor selection window. Since the events can only affect sensor state values in a short time interval, as described in Fig. 2, we use time windows to select informative pieces for event verification from the sensor data sequences.

We denote a time window as $(t^-_{E,S_i}, t^+_{E,S_i})$, which means for event (E, t_j, l_j) and sensor S_i, the window begins at $t_j + t^-_{E,S_i}$ and ends at $t_j + t^+_{E,S_i}$.

Since the baseline value of the sensor data fluctuates over time and physical environment (for example, the value of the air pressure sensor is relevant to the ambient atmosphere), we compute features from the relative change of sensor readings between windows, which is calculated by subtracting the sensor data in the current window by the previous one. Denote the current event timestamp as t_j and the previous event timestamp as t_{j-1}, the current window and the previous window are denoted as $(t_j + t^-_{E,S_i}, t_j + t^+_{E,S_i})$, $(t_{j-1} + t^-_{E,S_i}, t_{j-1} + t^+_{E,S_i})$, respectively. For the sensor data sequences $S_i(t)$, the sensor relative change is $S_i(t_1) - \overline{S_i(t_0)}$, where $t_1 \in (t_j + t^-_{E,S_i}, t_j + t^+_{E,S_i})$ and $t_0 \in (t_{j-1} + t^-_{E,S}, t_{j-1} + t^+_{E,S})$, in which $\overline{\bullet}$ means the average value of a sequence.

As each event-sensor pair has its own data change pattern, to determine a proper time window for each event-sensor pair is critical for feature extraction. We traverse and select the best time window from the candidates. We use Relative Mutual Information (RMI) as the metrics, which represents the the ratio of information provided by the sensors to the total information of an event.

To compute RMI, specifically, for an event E, a sensor S_i, and a window candidate $(t^-_{E,S_i}, t^+_{E,S_i})$, we first add window to the raw sensor data at each timestamp t_j, then compute features from the sequence in the corresponding time window $(t_j + t^-_{E,S_i}, t_j + t^+_{E,S_i})$. We compute five features for each sequence, which are maximum, minimum, sum, average, and standard deviation. The five features are aggregated into a feature vector denoted as $\vec{F_{S_i}}(t_j) = \{f_{S_i,1}(t_j), f_{S_i,2}(t_j), f_{S_i,3}(t_j), f_{S_i,4}(t_j), f_{S_i,5}(t_j)\}$.

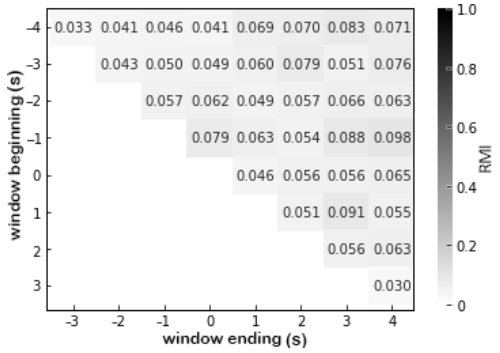
$RMI(E; S_i)$ between event E and sensor S_i is calculated as follows:

$$RMI(E; S_i) = \max(RMI(E; f_{S_i,m})) \quad (1)$$

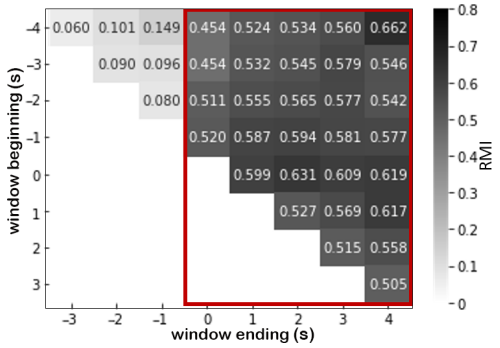
where $RMI(E; f_{S_i,m}) = \frac{I(l; f_{S_i,m})}{H(l)}$, and $m \in \{1, 2, 3, 4, 5\}$.

The maximum of all features is taken as the metric for this candidate window. We take all features $f_{S_i,m}(t)$ into consideration, computing the maximum value of $RMI(E; f_{S_i,m})$ for all $m \in \{1, 2, 3, 4, 5\}$. $RMI(E; f_{S_i,m})$ represents the ratio of information provided by feature $f_{S_i,m}$ to the total information of event E, which calculates from the event label sequence l and the corresponding feature sequence $f_{S_i,m}$. $I(\cdot)$ represents the mutual information between two sequence, and $H(\cdot)$ represents the entropy of a sequence.

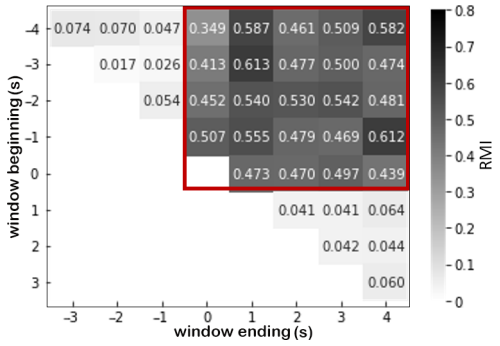
Figure 4 gives an example of sensor window selection. The example illustrates a grid search of events "Door opened" and "Door closed" with data from the accelerometer on the wall next to the door and from the accelerometer on the door. Figure 4 shows that the RMI of the accelerometer on the wall with "Door opened" is lower, and the RMI of "Door closed" is higher. This is because the physical influence between the door and the wall is usually smaller when opening the door than closing it. On the other hand, The RMI of the accelerometer on the door with "Door opened"



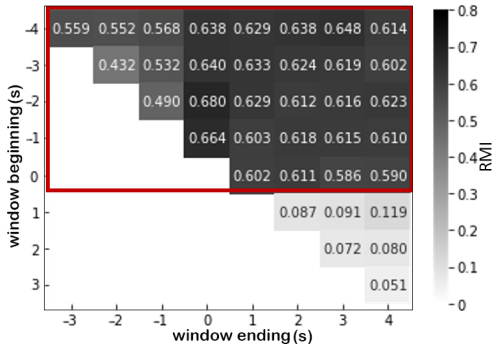
(a) Door opened vs. Accelerometer on the wall



(b) Door opened vs. Accelerometer on the door



(c) Door closed vs. Accelerometer on the wall



(d) Door closed vs. accelerometer on the door

Fig. 4 Typical result of sensor window grid search.

is higher when t^+ is greater than 0, that is, when the end of the time window is after the moment the "Door opened" event happens. It is obvious that the door itself is not accelerated before it is opened, so only under the condition that the end of the time window is after the event occurs, the RMI can get a high value. Similarly, the RMI of "Door closed" event and data from the sensor on the door are higher when t^- is less than 0, because the door does not move after closure. These phenomena are following real world situations and illustrate the rationality of the sensor window selection method as well.

(2) Feature computing. This sub-component computes all features from event timestamp, raw sensor data, and selected time windows from sensor window selection. After obtaining the optimal time windows, we compute features based on the aforementioned sensor relative variation. For each event (E, t_j, l_j), we take maximum, minimum, sum, average, and standard deviation as features in each window $(t_j + t^-_{E,S}, t_j + t^+_{E,S})$. Feature values are denoted as $f_{S_i,m}(t_j), m \in \{1, 2, 3, 4, 5\}$. The component computes these features for all sensors and outputs feature vector $\vec{F(t_j)}$.

4.3 Sensor selection

Sensor selection module is to determine the representative sensors for effective event validation and reduce the noise introduced by unrelated sensors. To achieve this, sensor selection uses a multi-layer perceptron Neural Network (NN) to analyze the importance of features, selects informative sensors as representative sensors and outputs a verification sensor set for event validation.

(1) Feature selection. Generally, layers with L1-regularization[30] can make the weight vector sparse (i.e., most elements of a vector are zero) to select features, and naturally avoid repeatedly selecting too many similar (or homogeneous) sensors. Thus by adding a one-to-one layer right after the input layer, the contribution of each feature input can be learned and reflected to corresponding weights. Such one-to-one weighting layer can select a more small subset of features comparing to traditional fully-connected networks[31]. In the proposed NN, the one-to-one layer is deployed to extract/output a sparse result after training, in which a 0 weight means that this feature has no contribution to the NN classifier and should be discarded.

Our proposed NN is illustrated in Fig. 5. The network includes a one-to-one layer, several hidden fully-connected layers, and one output node. The network takes all features as input and event labels as supervision (output). In the proposed network, the input feature

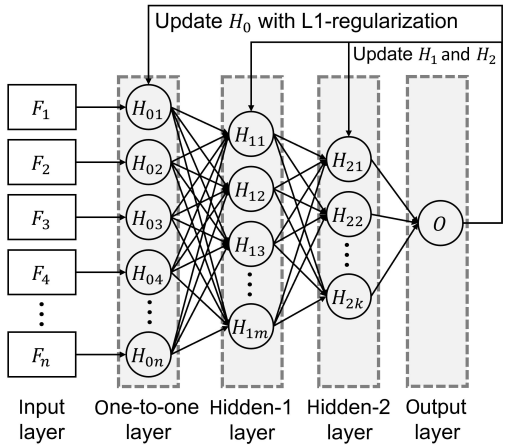


Fig. 5 Structure of the feature selection neural network.

vector is denoted as $\vec{F}$, the weight vector of the one-to-one layer is denoted as $\vec{\omega}$, and the output of this layer is denoted as $\vec{F} \circ \vec{\omega}$, where $\circ$ represents Hadamard product, i.e., element-wise multiplication. The output layer uses a sigmoid activate function to get a result between 0 and 1, and the rest layers use the ReLU function.

Empirically, the weight vector can hardly achieve a sparse result. The values will be very close to zero. This is due to the computing error introduced from gradient estimation. To solve this problem, we use an improved optimizer "SGD-L1(clipping)"[32,33]. The idea is to limit the weight value to cross zero during back-propagation, in order to avoid the weight oscillating near zero and enhance the sparseness.

The training process of our neural network consists of two hidden layers and an output layer. We denote the weight vector of one-to-one layer as $\vec{\omega}$, the weight vectors of other layers as $\vec{\omega_1}, \vec{\omega_2}$, and $\vec{\omega_3}$, and the bias as $\vec{b_1}, \vec{b_2}$, and $\vec{b_2}$. Note that all vectors aforementioned are column vectors in default.

During forward propagation, the input vector $\vec{x}$ is weighted by $\vec{\omega}$, and is denoted as $\vec{x_0} = \vec{x} \circ \vec{\omega}$. the weighted vector $\vec{x_0}$ is put into a fully-connected multi-layer network. We denote the loss function as follows:

$$J = l(y_{true}, y) + \lambda \|\vec{\omega}\|_1 \tag{2}$$

where y_{true} is the ground truth label, J is the sum of a criterion $l(\cdot)$ that measures the prediction error and L1-regularization that is the weight $\vec{\omega}$ times coefficient λ.

The neural network parameter update procedure is listed in Algorithm 1. In update algorithm, we use back-propagation to compute the gradient for each parameter in the neural network. We compute parameter weight after update using separated learning rates for the one-to-one layer and the other layers. Finally, for the one-to-one layer, we verify whether the sign of weight changes during the update, and if so we set its value to be 0.

Algorithm 1 Neural network parameter update procedure

Input: Model parameters from the last iteration: $\vec{\omega}$, $\vec{\omega_1}$, $\vec{\omega_2}$, $\vec{\omega_3}$, $\vec{b_1}$, $\vec{b_2}$, $\vec{b_3}$, loss J, and learning rate η_0 and η_1

Output: Model parameters after update $\vec{\omega'}, \vec{\omega_1'}, \vec{\omega_2'}, \vec{\omega_3'}, \vec{b_1'}$, $\vec{b_2'}$, and $\vec{b_3'}$

```
for each parameter p ∈ (ω, ω1, ω2, ω3, b1, b2, b3) do
    compute gradient δJ/δp using back-propagation;
end
// parameters excluding one-to-one layer
for each parameter p ∈ (ω1, ω2, ω3, b1, b2, b3) do
    p' = p − η1 δJ/δp;
end
// parameters in one-to-one layer, using SGD-L1(clipping)
for each parameter p ∈ ω do
    p' = p − η0 δJ/δp;
    // limit the parameter sign changes
    if (p × p' < 0) then
        p' = 0;
    end
end
```

(2) Sensor selection. The selected verification sensor set consists of the sensors that possess all non-zero weights. After we obtain the verification sensor set, the NN model is no longer useful that can be discarded.

4.4 Fingerprint generating, training, and checking

These modules use selected verification sensor set from above as the event fingerprint, and verify the event authenticity from the fingerprint by training classifiers.

(1) Offline training. In this phase, for each event tuple (E, t_j, l_j), we collect readings from the verification sensor set corresponding to the event near the timestamp t_j and generate the feature vector $F(\vec{t_j})$ by the feature construction module. We train binary support vector machine (SVM) classifiers to validate the authenticity of events. In the training phase, we use the feature vector $F(\vec{t_j})$ as the input and l_j as the supervised learning label (1 represents the event did occur and 0 represents the event did not occur). We apply different penalty parameters for two classes to deal with the situation that samples are imbalanced. Also, we do normalization for all samples to reduce the convergence time.

(2) Online checking. After feature construction and sensor selection modules are well pre-trained, the window selection and sensor selection results can be directly used for online preprocessing and checking.

In this phase, when we receive an event E^{test} for testing at the timestamp t^{test}, we collect its verification sensor data, generate its feature vector $F(\vec{t^{test}})$ by the fingerprint generating module and input them to the classifiers (double line arrow in Fig. 3). The output "0" represents that the testing event E^{test} is a forge event. The system only takes a small set of sensors as input and uses the pre-trained event verification classifiers, which reduces the complexity of online checking significantly.

5 Implementation

To evaluate our approach, we implement EScope and another existing solution, Peeves for comparison. In this section, we present the implemental design of EScope.

(1) Dataset. We used the dataset collected and shared by Birnbach et al. of Oxford University in their study[13]. This dataset, collected by a laptop and 12 Raspberry Pis (树莓派加s) in an office in 13 days, includes 49 sets of sensor data and 22 sets of event records. To ensure the accuracy, we conduct data cleaning and remove the abnormal/corrupted data, such as a continuous outlier over a long period of time and device failure.

(2) Preprocessing. Preprocessing aims to create three sub-datasets and generate sample labels from event records. As described in Section 4.2, we manually add negative samples at intervals on the time axis between positive samples. The total amount of samples is about 10 000. We separate it into three parts: the development set, the training set, and the testing set. The development set contains data collected in 1 day (i.e., 1/13 of the whole dataset) and is used to select time windows. As for data splitting of the training and the testing set, we conduct an experiment varying in different splitting proportions. The experiment shows no significant accuracy changes for the SVM when the proportion of the training set ranges from 40% to 80%. Real events rarely occur in the dataset (for example, the event "Radiator on" occurs only 18 times), and the amount of 1-samples is very small. Therefore, other splitting strategies (for example, 80% training/20% testing or 70% training/30% testing) will cause insufficient 1-samples in the testing set. Considering the sufficiency of 1-samples in the testing set and the minimization of the training time, we select 60% of the remaining 12 days as the training set and the other 40% as the testing set.

(3) Sensor window selection. The time window is the interval (t^-, t^+), with t^- ranging from $[-4, 3]$, and t^+ ranging from $[t^- + 1, 4]$. We traverse 36 windows in the range and select the window with the largest RMI of all features.

(4) RMI calculation. We calculate the mutual information between events and sensor features and divide mutual information by the information entropy of the event to get RMI. Generally, mutual information is calculated as described in the following:

$$I(A;B) = \sum_{b\in B}\sum_{a\in A} p(a,b)\log\left(\frac{p(a,b)}{p(a)\,p(b)}\right) \quad (3)$$

when random variables are discretely distributed. Yet, in practical, most sensors feature values follow continuous distribution.

Peeves uses a k-nearest-neighbors-based method to calculate the discrete-continuous mutual information. However, the testing result shows the method causes a high false rate in calculating the discrete-discrete mutual information. To process the discrete data more adequately, we divide them into bins and use the statistical frequency instead of probability to calculate the mutual information. We also evaluate another accurate estimation algorithm of mutual information for arbitrarily distributed data[35], which can be applied to the case of discrete and continuous mixing of eigenvalues of this work. However, based on our experiment, the time cost of the algorithm is unacceptable for our state correlation-based sensor selection.

(5) Sensor selection. We implement our neural network by PyTorch[36]. The number of the input nodes, i.e., the number of features, is 1340. We use a two-hidden-layer structure that has 400 and 100 nodes, respectively. Stochastic Gradient Descent (SGD) is used as the optimizer and Binary Cross Entropy (BCE) is used for the criterion, and we adjust it by adding the L1-norm of one-to-one layer. The loss function is $J = BCE(y_{true}, y) + \lambda\|\vec{\omega}\|_1$, where λ is the penalty of the L1-regularization and $\vec{\omega}$ is the weight vector of the one-to-one layer. This loss function will adjust the high value of the weight vector to get a sparse result. Furthermore, by adjusting λ we can change the degree of sparseness and control the number of selected features, and then control the size of verification sensor sets.

As for hyper-parameters, we tried different optimizers and learning rates. We found that changing optimizers has little impact on convergence performance. We adjust the learning rate of one-to-one layer separately for a faster weight vector update. According to our experiment, setting a higher learning rate of one-to-

one layer compared to other layers gains a more sparse result. Since we only need the weight vector, we only consider whether the loss is stable and well-reduced during training, which means the model converges.

(6) SVM classifier. We use scikit-learn[37] in Python to build the SVM classifier. The $class\_weight$ parameter is introduced to apply different penalty Cs in two classes of samples, so as to avoid the deviation of the classification boundary caused by the imbalance of the sample numbers. According to our tests, changing C has little effect on classification performances, and changing $class\_weight$ has a slight impact on the model test results: the results improve as $class\_weight$ increases from 1, however, after reaching a certain threshold, not only the model test results hardly improve with the increase of $class\_weight$, but the training time cost increases.

(7) Parameter tuning. To determine NN hyperparameters (including learning rate and λ), we traverse all candidates and deal with the trade-off between performance (F1-score) and sensor amounts. In detail, we set an F1-score baseline at 0.95, and a sensor amount baseline at 10. We denote the score as $s = |\text{F1-score} - 0.95|^{1.5} + |amount_s - 10|$, where $anount_s$ represents the amount of selected sensors, this encourages high F1-scores and low sensor amounts, and pick the hyperparameters with the highest score. We use grid search to adjust the parameters in our SVM (including the kernel, C, and $class\_weight$) for each event separately, in order to get the best performances.

(8) Online checking deployment. As the event authentication mechanism, the online checking component might be deployed in the IoT control system, following the event notification receiver component. When the event notification receiver passes an event notification to the online checking component, it will collect the selected representative sensors' data, generate the feature vector as the classifiers' input, and output the verification result. If the result is "0", the event will be treated as a forged one that cannot pass through the authentication and will be intercepted before triggering succeeding actions.

6 Evaluation and Discussion

In this section, we conduct experiments to evaluate the performance of EScope and illustrate the advantages of our approach by answering the following questions:

- **Q1:** Is EScope effective to detect forged events?
- **Q2:** Does EScope manage to build more accurate event fingerprints?
- **Q3:** Is our NN-based sensor selection method better compared to threshold-based methods?
- **Q4:** What is the influence of neural network parameters on sensor amounts?

We re-implement the prior work, Peeves[13], on the same dataset for comparison. We follow the same procedure in Peeves, in which it uses RMI thresholds to select features, and set 40% by default, to limit the noisy features and reduce the time burden.

6.1 Forged event detection performance

To answer question Q1, we replay event spoofing attacks by simulation and evaluate the detection performance of EScope.

(1) Dataset preparation. The dataset we used includes event records and sensor data. We add 20 items randomly in time to the event record as forged events, meanwhile remaining the sensor data unchanged, so that the system believes the event did happen but the sensor data have no reaction, which is the situation of event spoofing attacks. When adding forged events, we set a safety margin (10 seconds) to avoid overlapping with real events.

(2) Event verification. We search for sensors in the verification sensor set given by the sensor selection module in our approach, then we compute features on these sensors to build learning samples, and finally, we use our pre-trained SVM classifier to give predictions, which simulates the online running process of EScope.

We reserve all positive events from the testing set, i.e., the real events. Then we discard the negative events that indicate the event did not occur and use our injected forged events as negative. This can indicate the ability to detect potential event spoofing attacks from IoT event records.

We also implement our simulated attacks on Peeves[13] and evaluate their detection accuracy. As for the metrics, we compute F1-score $= 2 \times \dfrac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$ from the prediction result from the model, as shown in the left part of Table 3. We also list the number of sensors for event verification in Table 3.

From the result in Table 3, we can see that for 17 of the 22 events in total, EScope obtains an F1-score larger than 0.9, and the F1-score of all events are larger than 0.75, which indicates that our method is effective enough to detect potential event spoofing attacks. We also compare our approach with EScope, and it is shown

Table 3 Performance and number of selected sensors of different approaches.

Event name	Evaluation results in Section 6.1		Evaluation results in Section 6.2				Evaluation results in Section 6.3		Number of sensors			
	F1-score		F1-score		AUC		F1-score					
	Peeves	EScope	Peeves	EScope	Peeves	EScope	MS	Peeves+MS	Peeves	EScope	MS	Peeves+MS
Window opened	0.9973	**1.0000**	0.9665	**1.0000**	1.0000	**1.0000**	0.7112	0.7749	32	**3**	8	8
Light off	0.9938	**1.0000**	1.0000	0.9442	1.0000	**1.0000**	0.9058	0.8325	27	**2**	10	10
Door closed	0.9996	0.9977	1.0000	0.9979	1.0000	**1.0000**	0.6740	0.9980	25	**2**	9	9
Fridge opened	0.9893	**0.9893**	0.9883	0.9217	1.0000	0.9982	0.9415	0.6452	29	**7**	11	11
PC on	0.9634	**0.9881**	0.9842	**0.9897**	0.9624	**0.9999**	0.4989	0.4988	3	5	10	10
Fridge closed	1.0000	0.9872	0.9883	**0.9893**	1.0000	0.9925	0.5897	0.7485	29	**6**	12	12
Door opened	0.9992	0.9858	0.9982	0.9974	1.0000	**1.0000**	0.9222	0.9505	20	**2**	11	11
Fan off	0.9805	**0.9842**	0.9500	**0.9616**	0.9908	**0.9968**	0.8734	0.9795	9	**2**	10	10
Window closed	0.9866	0.9839	0.9649	0.9088	0.9999	0.9995	0.5714	0.6084	25	**6**	9	9
Fan on	0.9395	**0.9804**	0.9238	**0.9746**	0.8878	**0.9989**	0.5762	0.7040	2	5	9	9
Coffee machine used	1.0000	0.9787	1.0000	**1.0000**	1.0000	**1.0000**	0.7146	0.8909	34	**5**	10	10
Light on	0.9969	0.9779	1.0000	0.9641	1.0000	**1.0000**	0.9883	1.0000	23	**5**	10	10
Screen off	0.9535	**0.9626**	0.8657	**0.9735**	0.8696	**0.9875**	0.8651	0.8662	3	5	10	10
Screen on	0.9719	0.9618	0.9004	**0.9537**	0.9514	**0.9577**	0.5318	0.8167	9	**4**	11	11
Camera on	0.9533	**0.9542**	0.7925	**0.8248**	0.9046	**0.9805**	0.5324	0.8661	11	**7**	11	11
Camera off	0.9180	**0.9268**	0.7648	**0.8223**	0.8997	**0.9340**	0.6232	0.6196	10	14	12	12
Shade up	0.7918	**0.9013**	0.4761	**0.9472**	0.5479	**0.9789**	0.4481	0.4682	13	**6**	9	9
Shade down	0.7660	**0.8928**	0.5032	**0.9140**	0.5820	**0.9703**	0.4985	0.4985	10	**9**	12	12
PC off	0.9233	0.8621	0.7269	**0.9665**	0.8539	**1.0000**	0.7538	0.5219	4	5	12	12
Radiator on	0.7552	**0.8303**	0.4894	**0.8473**	0.4687	**0.8180**	0.5207	0.4992	16	**8**	10	10
Doorbell used	0.8460	0.7927	0.4472	**0.8095**	0.7316	**0.8923**	0.5825	0.5335	19	**5**	11	11
Radiator off	0.7172	**0.7774**	0.2123	**0.5636**	0.4287	**0.8664**	0.4636	0.5408	10	**7**	9	9
Average	0.9292	**0.9416**	0.8156	**0.9214**	0.8672	**0.9714**	0.6721	0.7210	16.5	**5.45**	10.3	10.3

Note: MS denotes manual selection.

that EScope has an average F1-score improvement to Peeves. In addition, we illustrate the number of sensors used for event verification in Table 3. As shown in Table 3, EScope uses less sensors (5.45 compared to 16.50 on average) than Peeves. It means that EScope uses a smaller verification sensor set to achieve a relatively good validation accuracy. In 14 cases, EScope has a better detection accuracy (marked in bold).

Answer to Q1: EScope obtains an average F1-score of 0.9416 to detect forged events and exceeds Peeves by 0.0124, which demonstrates the effectiveness of EScope in forged event detection.

6.2 Event fingerprint performance

To answer question Q2, we use several metrics (such as F1-score and AUC) to measure the event fitting performance of EScope. The evaluation is based on the original dataset from Peeves[13]. Testing set split from preprocessing is directly used for evaluation, which includes positive standing for real events and negative standing for not happened events. The result indicates the accuracy of event fingerprints for event occurrence detection and shows the inherent ability to fit the physical environment rules of event fingerprint models.

We calculate F1-score $= 2 \times \dfrac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$ of the classifier for both positive and negative samples and take their average. We plot the corresponding Receiver Operator Characteristic (ROC) curve, which takes False Positive Rate (FPR) and True Positive Rate (TPR) as the axis. For the ROC, the curve closer to the upper left corner is considered to be a better result, which has a high TPR when FPR is small, and meanwhile, this curve has a larger AUC. F1-score is more rational under the condition of the numbers of positive and negative samples are imbalanced, and AUC focuses on the intrinsic performance of the classifier because it takes the decision threshold into account.

We show the F1-score, AUC and number of sensors comparison of two event fingerprint models in the right part of Table 3, which is calculated from the classification results output from the SVM classifier. We can see that 17 events have F1-scores larger than 0.9 in EScope. It demonstrates that EScope generates effective

event fingerprint models through sensor data.

Compared to the existing solution, Peeves lags behind EScope in both F1-score and AUC. Specifically, our method has significant improvements on "Radiator on", "Radiator off", "Doorbell used", "Shade up", and "Shade down" events. This is facilitated by our proposed NN-based approach that outputs highly-informative verification sensor sets by discovering implicit and complex relationships between events and sensors. The results demonstrate that EScope generates more accurate event fingerprints with a much smaller verification sensor set.

We plot the ROC curve of three events (Fig. 6), from which we can see that the curve of EScope is closer to the upper left corner, which indicates that our approach has a better performance.

Answer to Q2: EScope obtains an average F1-score of 0.9214 and AUC of 0.9714 in the event fingerprint generation experiment, which exceeds Peeves by 0.1058 and 0.1042. It indicates that EScope can create more accurate event fingerprints.

6.3 Case studies of sensor selection

To answer question Q3, we deploy two cases to illustrate the performance of our sensor selection method. We compute inter-feature correlation coefficients of each other, we take the maximum of the features corresponding to the sensors. We show the heat maps in Fig. 7, which illustrates the absolute values of the correlation coefficients between sensors. Both x-axis and y-axis represent sensors. Red means the correlation coefficient absolute value between these two sensors is closed to 1, and blue means the value is closed to 0. We illustrate the sensor selection result (i.e., the verification sensor set) and classification metric (AUC) in Tables 4 and 5. We cluster sensors according to their correlation coefficients by clustering, where $1-|corr_{i,j}|$ is considered as the distance between sensors i and j, and hierarchical clustering is conducted according to the

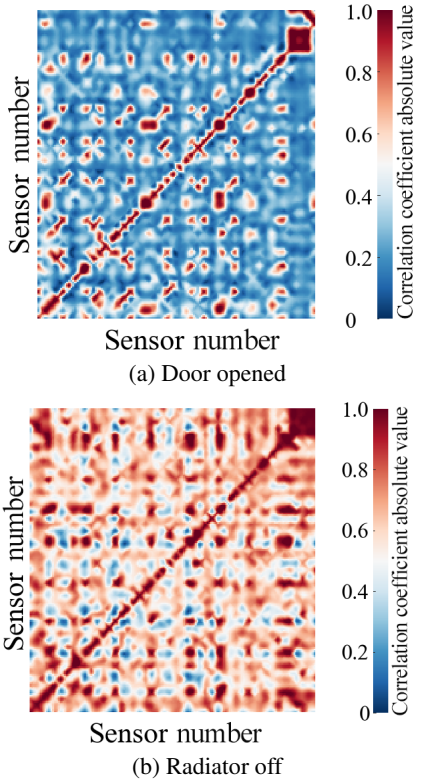


Fig. 7 Correlation heat-maps between sensors.

distance. The sensors in the same cluster provide similar information to event fingerprints, which is homogeneous.

Case study 1: Event "Door opened". Figure 7a shows that for the "Door opened" event, the inter-sensor correlation is relatively low, while there still are some deep color dots, which means highly-correlated sensor pairs. The clustering result is shown in Table 4. In this case, our approach avoids selecting multiple sensors in one cluster (Clasters A and B), and discards other clusters selected by Peeves.

Specifically, EScope only selects two sensors for the event "Door opened", which are pi7_BME680 (temperature, humidity, and air pressure sensor beside the door) and pi8_SenseHat (acceleration sensor on the door). Since the door opening action affects the

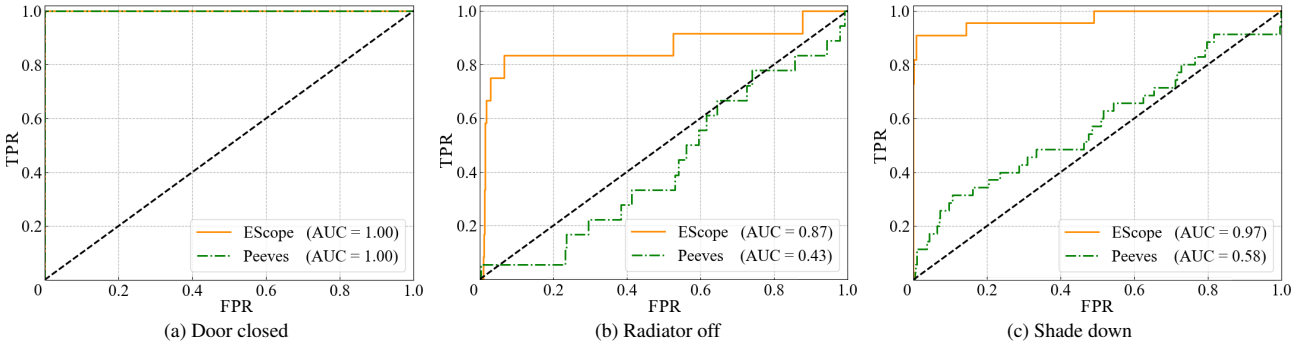


Fig. 6 ROC curves of three events.

acceleration and air pressure, these two sensors are both highly informative for this event. Table 4 also shows that merely using these two sensors can achieve an AUC of 1, which demonstrates that EScope selects more representative sensors compared to Peeves.

Case study 2: Event "Radiator off". For the "Radiator off" event in Fig. 7b, the inter-sensor correlation is high. Accordingly, Table 5 shows that our approach selects completely different sensor sets, and significantly improves the AUC. Our approach achieves better performances in choosing more informative sensors.

In this case study, most sensors selected by EScope are heat-related (for example, MLX represents for thermal camera, MPU6050 senses accelerator and temperature). However, most sensors selected by Peeves are irrelevant. Thus, EScope can select more informative sensors.

Additionally, in order to prove the necessity to use neural networks instead of intuitively choosing sensors according to their correlation, we perform two manual sensor selection, experiments, which are ① cluster all sensors according to correlation, then randomly pick one from each cluster, and ② use Peeves sensor selection method, cluster sensors and then randomly pick one from each cluster. ① focuses on getting rid of choosing homogeneous sensors and ② does a similar job based on the Peeves sensor selection result. The classification result is shown in Table 3. The result shows that both two approaches can reduce the sensor amount and obtain a high F1-score on several events such as "Light on", but on most events, EScope performs much better.

Answer to Q3: Compared to the existing threshold-based sensor selection method Peeves, EScope selects representative sensors intelligently, which reduces the verification sensor amount while improving accuracy. Also, our method performs better than manual sensor selection.

6.4 Parameter influence analysis

To answer question Q4, we adjust the number of sensors to observe the AUC changes. We evaluate our approach and Peeves for comparison. For our approach, we adjust the L1 penalty coefficient λ to control the number of sensors. We show the number of sensors change according to λ at Fig. 8. Due to the unsteadiness of neural networks, for each point on the graph, we train the networks several times and take the average.

Generally, the number of sensors decreases as the L1 penalty λ increases. There are several outliers at high λ value of event "Camera off" (in red rectangle), which is not an accident confirmed in our further experiment, because high λ reduces the importance of error between predict value and ground truth, causing loss unable to fall, and making the NN fail to work. In our evaluation,

Table 4 Door opened verification sensor sets.

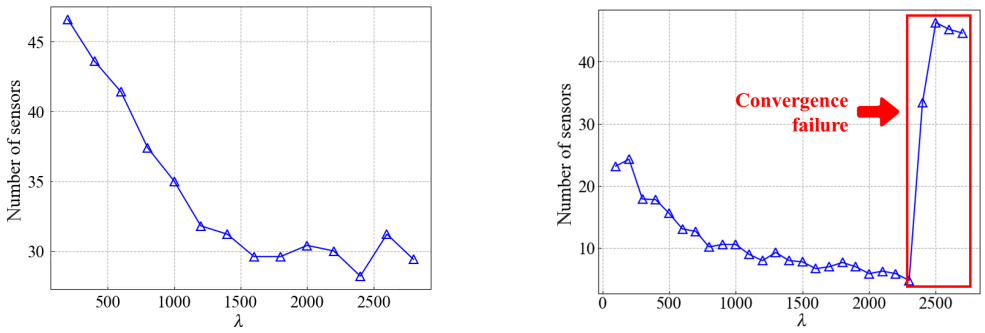
Method	AUC	Cluster	Verification sensor set
EScope	1.0000	Cluster A	pi7_BME680
		Cluster B	pi8_SenseHat
Peeves	1.0000	Cluster A	pi10_BME680_IN, pi10_BME680_OUT
			pi1_microphone, pi2_microphone, pi3_microphone
			pi5_microphone, pi7_microphone, pi9_microphone
		Cluster B	pi3_BME680, pi7_BME680
			pi1_BMP280, pi2_BMP280, pi5_BMP280
			pi6_BMP280, pi8_SenseHat
		Cluster C	pi1_TSL2560, pi7_TSL2560
		Cluster D	pi1_RSS
		Cluster E	pi7_RSS

Note: The number behind "pi" represents the Raspberry pi ID during data collecting.
BME280/BMP280/BME680 — Sensor module for temperature/humidity/pressurem, respectively; SenseHat — Sensor module for temperature, humidity, pressure, and acceleration; TSL2560 — Sensor module for illuminance; RSS — Received Signal Strength;

Table 5 Radiator off verification sensor sets.

Method	AUC	Verification sensor set
EScope	0.8664	pi11_MLX90640, pi12_MLX90640, pi3_MPU6050
		CamPower, pi4_MPU6050, Pi5_TSL2560, pi9_MPU6050
Peeves	0.4287	pi5_RSS, PCPower, pi10_BME680_IN
		pi1_BMP280, pi2_RSS, pi6_BMP280
		FanPower, ScreenPower, pi7_RSS, WindowShadePower

Note: MLX90640 — Thermal camera; MPU6050 — Sensor module for accelerator and temperature.



(a) Camera on (b) Camera off

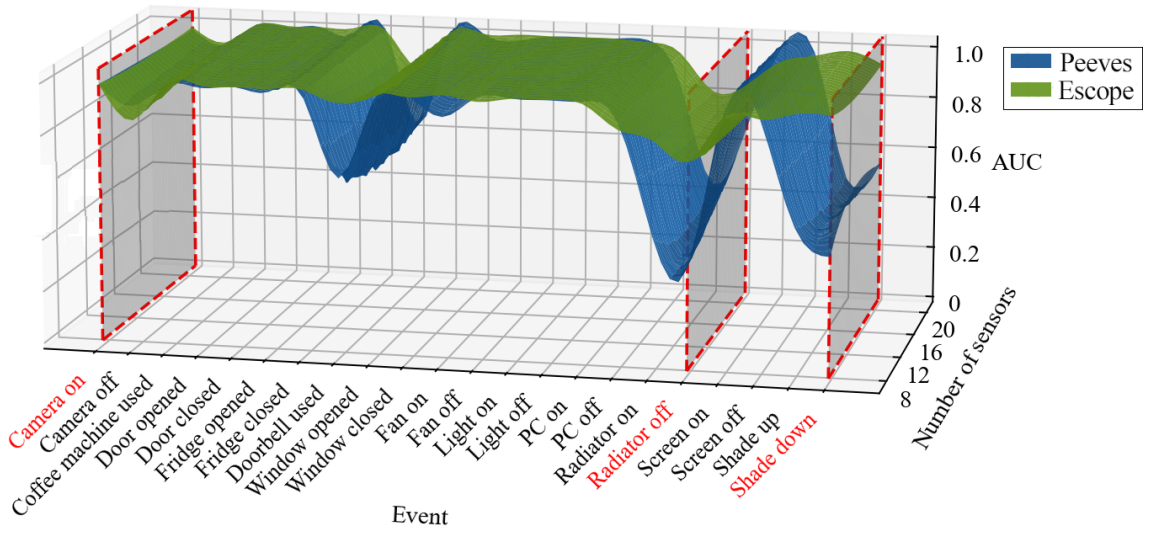
Fig. 8 Influence of λ on the number of sensors.

we avoid these outliers by limiting λ.

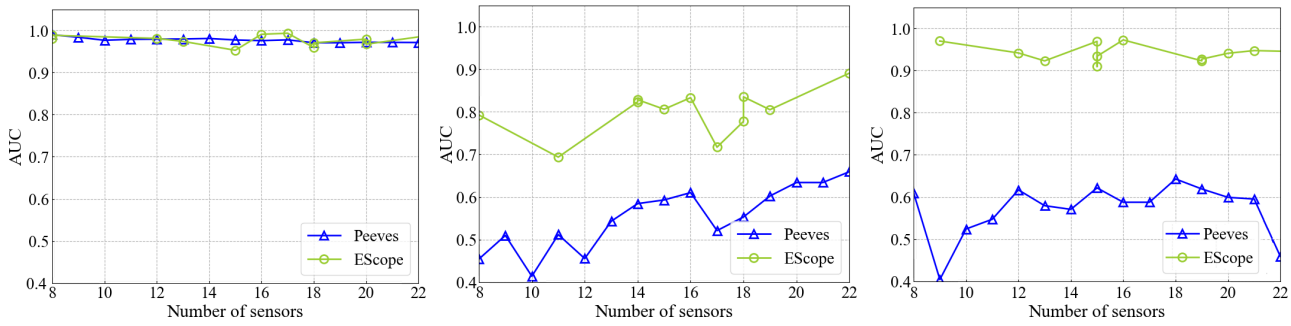
To evaluate the parameter influence in Peeves, we change the RMI threshold, as described in Section 5, to adjust the number of sensors.

Figure 9a illustrates the relationship between AUC and the number of sensors at each event using 3D surfaces. As shown in Fig. 9a, for most events, the green surface is above the yellow surface, which means that our proposal can obtain better performance using the same amount of sensors. To illustrate elaborately, we show several event cross-sections in Fig. 9a, specialized in Figs. 9b–9d. It can be found that for the event "Camera on", there is little difference in the effectiveness of the two methods, while for the event "Radiator off" and "Shade down", our approach outperforms Peeves.

Answer to Q4: The L1-regularization penalty λ and the number of selected sensors are negatively correlated. Additionally, EScope outperforms Peeves in AUC when the sensor amount of the two methods are identical.



(a) 3D graph of AUC vs. the number of sensors of all events



(b) Camera on (c) Radiator off (d) Shade down

Fig. 9 Relationship between AUC and number of sensors.

7 Conclusion

In this paper, we propose EScope, an effective event validation approach to check the authenticity of system events based on device state correlation. EScope filters informative and efficient sensors using an NN-based sensor selection component and extracts a verification sensor set for event validation. We prototyped our approach and conducted evaluations on a public data set provided by Peeves. The experiment results show that our approach has better validation performance on F1-score and AUC compared with existing solutions, and significantly reduces the number of necessary verification sensors. We also demonstrate that our NN-based sensor selection method is more effective to deal with highly-correlated features.

Acknowledgment

The authors are very grateful to all reviewers who have helped improve the quality of this paper. This work was supported in part by the National Natural Science Foundation of China (Nos. 62172027, U1733115, and 61871023), the Beijing Natural Science Foundation (No. 4202036), and the National Key R&D Program of China (No. 2020YFB1005601).

References

[1] M. Azrour, J. Mabrouki, A. Guezzaz, and Y. Farhaoui, New enhanced authentication protocol for internet of things, *Big Data Mining and Analytics*, vol. 4, no. 1, pp. 1–9, 2021.

[2] J. Mabrouki, M. Azrour, D. Dhiba, Y. Farhaoui, and S. El Hajjaji, IoT-based data logger for weather monitoring using arduino-based wireless sensor networks with remote graphical application and alerts, *Big Data Mining and Analytics*, vol. 4, no. 1, pp. 25–32, 2021.

[3] J. Mabrouki, M. Azrour, G. Fattah, D. Dhiba, and S. El Hajjaji, Intelligent monitoring system for biogas detection based on the internet of things: Mohammedia, morocco city landfill case, *Big Data Mining and Analytics*, vol. 4, no. 1, pp. 10–17, 2021.

[4] Samsung, One simple home system. A world of possibilities, https://www.smartthings.com, 2022.

[5] HomeAssistant, Home assistant, https://www.homeassistant.io, 2022.

[6] C. Aranzazu-Suescun and M. Cardei, Distributed algorithms for event reporting in mobile-sink WSNs for internet of things, *Tsinghua Science and Technology*, vol. 22, no. 4, pp. 413–426, 2017.

[7] C. Fu, Q. Zeng, and X. Du, HAWatcher: Semantics-aware anomaly detection for appified smart homes, in *Proc. 30th USENIX Security Symp.*, Virtual Event, 2021, pp. 4223–4240.

[8] Samsung smart things, Mobile device presence update delay, https://community.smartthings.com/t/mobiledevice-presence-update-delay/98672, 2022.

[9] L. Russell, Wireless security monitoring versus a cellular jammer, https://www.home-security-systemsanswers.com/wireless-security-monitoring.html, 2022.

[10] E. Fernandes, J. Jung, and A. Prakash, Security analysis of emerging smart home applications, in *Proc. of the 2016 IEEE Symp. Security and Privacy* (*SP*), San Jose, CA, USA, 2016, pp. 636–654.

[11] Z. Cai and Z. He, Trading private range counting over big IoT data, in *Proc. of the 2019 IEEE 39th Int. Conf. Distributed Computing Systems* (*ICDCS*), Dallas, TX, USA, 2019, pp. 144–153.

[12] E. Ronen, A. Shamir, A. O. Weingarten, and C. O'Flynn, IoT goes nuclear: Creating a ZigBee chain reaction, in *Proc. of the 2017 IEEE Symp. Security and Privacy* (*SP*), San Jose, CA, USA, 2017, pp. 195–212.

[13] S. Birnbach, S. Eberz, and I. Martinovic, Peeves: Physical event verification in smart homes, in *Proc. 2019 ACM SIGSAC Conf. Computer and Communications Security*, London, UK, 2019, pp. 1455–1467.

[14] A. Acar, H. Fereidooni, T. Abera, A. K. Sikder, M. Miettinen, H. Aksu, M. Conti, A. R. Sadeghi, and S. Uluagac, Peek-a-boo: I see your smart home activities, even encrypted!, in *Proc. 13th ACM Conf. Security and Privacy in Wireless and Mobile Networks*, Linz, Austria, 2020, pp. 207–218.

[15] X. Zheng and Z. Cai, Privacy-preserved data sharing towards multiple parties in industrial IoTs, *IEEE J. Select. Areas Commun.*, vol. 38, no. 5, pp. 968–979, 2020.

[16] W. Zhang, Y. Meng, Y. Liu, X. Zhang, Y. Zhang, and H. Zhu, HoMonit: Monitoring smart home apps from encrypted traffic, in *Proc. 2018 ACM SIGSAC Conf. Computer and Communications Security*, Toronto, Canada, 2018, pp. 1074–1088.

[17] R. Trimananda, J. Varmarken, A. Markopoulou, and B. Demsky, Packet-level signatures for smart home devices, presented at the Network and Distributed Systems Security (NDSS) Symp., San Diego, CA, USA, 2020.

[18] G. Laput, Y. Zhang, and C. Harrison, Synthetic sensors: Towards general-purpose sensing, in *Proc. 2017 CHI Conf. Human Factors in Computing Systems*, Denver, CO, USA, 2017, pp. 3986–3999.

[19] Y. Liang, Z. Cai, J. Yu, Q. Han, and Y. Li, Deep learning based inference of private information using embedded sensors in smart devices, *IEEE Netw.*, vol. 32, no. 4, pp. 8–14, 2018.

[20] R. Yasaei, F. Hernandez, and M. A. Al Faruque, IoT-CAD: Context-aware adaptive anomaly detection in IoT systems through sensor association, in *Proc. of the 2020 IEEE/ACM Int. Conf. Computer Aided Design* (*ICCAD*), San Diego, CA, USA, 2020, pp. 1–9.

[21] A. Guezzaz, Y. Asimi, M. Azrour, and A. Asimi, Mathematical validation of proposed machine learning classifier for heterogeneous traffic and anomaly detection, *Big Data Mining and Analytics*, vol. 4, no. 1, pp. 18–24, 2021.

[22] Z. Cai and X. Zheng, A private and efficient mechanism for data uploading in smart cyber-physical systems, *IEEE Trans. Netw. Sci. Eng.*, vol. 7, no. 2, pp. 766–775, 2020.

[23] C. Debes, A. Merentitis, S. Sukhanov, M. Niessen, N. Frangiadakis, and A. Bauer, Monitoring activities of daily living in smart homes: Understanding human behavior, *IEEE Signal Process. Mag.*, vol. 33, no. 2, pp. 81–94, 2016.

[24] K. Islam, W. Shen, and X. Wang, Security and privacy considerations for wireless sensor networks in smart home environments, in *Proc. 2012 IEEE 16th Int. Conf. Computer Supported Cooperative Work in Design* (*CSCWD*), Wuhan, China, 2012, pp. 626–633.

[25] X. Zheng, Z. Cai, and Y. Li, Data linkage in smart internet of things systems: A consideration from a privacy perspective, *IEEE Commun. Mag.*, vol. 56, no. 9, pp. 55–61, 2018.

[26] V. Sivaraman, D. Chan, D. Earl, and R. Boreli, Smartphones attacking smart-homes, in *Proc. 9th ACM Conf. Security & Privacy in Wireless and Mobile Networks*, Darmstadt, Germany, 2016, pp. 195–200.

[27] T. Gu, Z. Fang, A. Abhishek, H. Fu, P. Hu, and P. Mohapatra, IoTGaze: IoT security enforcement via wireless context analysis, in *Proc. of the IEEE INFOCOM 2020-IEEE Conf. Computer Communications*, Toronto, Canada, 2020, pp. 884–893.

[28] J. Wang, S. Hao, R. Wen, B. Zhang, L. Zhang, H. Hu, and R. Lu, IoT-praetor: Undesired behaviors detection for IoT devices, *IEEE Int. Things J.*, vol. 8, no. 2, pp. 927–940, 2021.

[29] F. L. Coman, K. M. Malarski, M. N. Petersen, and S. Ruepp, Security issues in internet of things: Vulnerability analysis of LoRaWAN, Sigfox and NB-IoT, in *Proc. of the 2019 Global IoT Summit* (*GIoTS*), Aarhus, Denmark, 2019, pp. 1–6.

[30] R. Tibshirani, Regression shrinkage and selection via the lasso, *J. Roy. Stat. Soc.: Ser. B (Methodol.)*, vol. 58, no. 1, pp. 267–288, 1996.

[31] Y. Li, C. Y. Chen, and W. W. Wasserman, Deep feature selection: Theory and application to identify enhancers and promoters, *J. Comput. Biol.*, vol. 23, no. 5, pp. 322–336, 2016.

[32] B. Carpenter, Lazy sparse stochastic gradient descent for regularized multinomial logistic regression, Tech. Rep., Alias-i, Inc., New York, NY, USA, 2008, pp. 1–20.

[33] Y. Tsuruoka, J. Tsujii, and S. Ananiadou, Stochastic gradient descent training for L1-regularized log-linear models with cumulative penalty, in *Proc. Joint Conf. 47th Annu. Meeting of the ACL and the 4th Int. Joint Conf. Natural Language Processing of the AFNLP*, Suntec, Singapore, 2009, pp. 477–485.

[34] B. C. Ross, Mutual information between discrete and continuous datasets, *PLoS ONE*, vol. 9, no. 2, p. e87357, 2014.

[35] W. Gao, S. Kannan, S. Oh, and P. Viswanath, Estimating mutual information for discrete-continuous mixtures, arXiv preprint arXiv: 1709.06212, 2018.

[36] Facebook, Pytorch, https://pytorch.org, 2022.

[37] Sklearn, scikit-learn: Machine learning in python, https://scikit-learn.org/stable, 2022.

Jian Mao received the BEng and PhD degrees from Xidian University, China in 1997 and 2004, respectively. She is an associate professor at the School of Cyber Science and Technology, Beihang University, Beijing, China. Her research interests include IoT security, web security, mobile security, and social network security.

Xiaohe Xu received the BEng degree in electronic and information engineering from Beihang University, Beijing, China in 2020. Currently, he is a master student at the School of Cyber Science and Technology, Beihang University, Beijing, China. His research interests include learning-based security analysis and IoT security.

Qixiao Lin received the BEng degree in electronic and information engineering from Beihang University, Beijing, China in 2018. Currently, he is a PhD candidate at the School of Cyber Science and Technology, Beihang University, Beijing, China. His research interests include learning-based security analysis, social network privacy preserving, and IoT security.

Liran Ma received the PhD degree in computer science from the George Washington University, USA in 2008. He is currently a professor at the Department of Computer Science, Texas Christian University, Fort Worth, USA. His research interests include wireless networking, IoT, systems security, data analytics, and security education. He is a member of the IEEE.

Jianwei Liu received the BEng and MEng degrees in electronic and information engineering from Shandong University, China in 1985 and 1988, respectively. He received the PhD degree in communication and electronic system from Xidian University, China in 1998. Now, he is a professor at the School of Cyber Science and Technology, Beihang University, Beijing, China. His current research interests include wireless communication network, cryptography, and network and information security.