CP-IoT: A Cross-Platform Monitoring System for Smart Home

Hai Lin[1,2], Chenglong Li[1,2] ✉, Jiahai Yang[1,2] ✉, Zhiliang Wang[1,2]
Linna Fan[3], Chenxin Duan[1]
[1]Institute of Network Sciences and Cyberspace, BNRist, Tsinghua University
[2]Zhongguancun Laboratory, [3]National University of Defense Technology
{linhai22, fln19, dcx19}@mails.tsinghua.edu.cn
{lichenglong}@tsinghua.edu.cn, {wzl, yjh}@cernet.edu.cn

***Abstract*—Today, smart home platforms are widely used around the world and offer users automation to define their daily routines. However, individual automation rule anomalies and cross-automation threats that exist in different platforms put the smart home in danger. Recent researches focus on detecting these threats of the specific platform and can only cover limited threat plane. To solve these problems, we design a novel system called CP-IoT, which can monitor the execution behavior of the automation and discover the anomalies, as well as hidden risks among them on heterogeneous IoT platforms. Specifically, CP-IoT constructs a centralized, dynamic graph model for portraying the behavior of automation and the state transition. By analyzing two kinds of app pages with different description granularity, CP-IoT extracts the rule execution logic and collects user policy from different platforms. To detect the inconsistent behavior of an automation rule in different platforms, we propose a self-learning method for event fingerprint extraction by clustering the traffic of different platforms collected from the side channel, and an anomaly detection method by checking the rule execution behavior with its specification reflected in the graph model. To detect the cross-rule threats, we formalize each threat type as a symbolic representation and apply the searching algorithm on the graph. We validate the performance of CP-IoT on four platforms. The evaluation shows that CP-IoT can detect anomalies with high accuracy and effectively discover various types of cross-rule threats.**

I. Introduction

With the rapid development of computer technologies such as embedded systems and wireless communications, IoT (Internet of Things) devices have been widely deployed all around the world. The smart home has been enabled after manufacturers produce various kinds of smart IoT devices. Nowadays, smart home platforms including Amazon Alexa [1], Samsung SmartThings [2], Apple Homekit [3], Google Home [4] and Xiaomi Home [5] dominates the global market. They provide users with home automation (HA) to define their daily routine such as turning on the camera and locking the door after the user leaves home. Some researches [6], [7] have shown that most applications and automations can be represented by the trigger-action programming (TAP) model or "If-This-Then-That" (IFTTT) programming paradigm/rule.

However, security issues about automations are discussed daily in various communities [8]–[10]. For example, an anomaly case is reported in SmartThings Community, where sensors detect user activity without triggering the rule [11]. A study [12] finds 76 instances of 222 SmartThings apps are over-privilege, with 5.5% of related devices being misused. And a recent study [13] shows that more than 55% of SmartApps have extra permission to control unrelated devices. To solve these problems, Homonit [14] and IoTGaze [15] perform side-channel inference on encrypted traffic to obtain the execution behavior of the automation rule and match it with the rule specification extracted from SmartApps. HAWatcher [16] collects various semantic information from the smart home and constructs correlations between each event and device. It takes a similar way to detect violations of these correlations by checking the runtime behavior of automations.

Apart from the execution faulty in individual automation rules, the security problem also exists among different automation rules. Considering two examples: (1) A rule turns on the fan when the sensor detects the motion, which reduces the temperature and unexpectedly triggers another rule to turn on the heater when the sensor detects the low temperature. (2) A rule turns on the light when detects the motion and another rule turns off the light when the user is at home, which results in an action conflict. Soteria [17] and IoTGuard [18] collect information from SmartApp to build an execution flow graph of automation rules as their dynamic model and check whether the combination of the rules violates the security properties to detect some of the two threats mentioned above. iRuler [19] refers to (1) as implicit and explicit chaining and (2) as inter-rule vulnerability, while Homeguard [20] refers to both uniformly as cross-app interference (CAI). However they are all defined for the SmartThings SmartApp. Since applications on different platforms can be abstracted into automation rules, we unify the above two kinds as cross-automation threats, where (1) is denoted as cross-rule interaction and (2) is denoted as cross-rule interference.

The above researches have greatly enhanced the security and safety of the HA systems, but have the following two limitations: 1) Limited threat detection plane. They detect inconsistent behavior of individual automation rules or cross-rule threats, but not both. Moreover, their detection results are inadequate since they neglect to detect some fine-grained subtypes. 2) Low platform compatibility. Most of them are

✉ Corresponding authors.

Network and Distributed System Security (NDSS) Symposium 2024
26 February - 1 March 2024, San Diego, CA, USA
ISBN 1-891562-93-2
https://dx.doi.org/10.14722/ndss.2024.24003
www.ndss-symposium.org

designed for SmartThings and develop code-based methods both for rule extraction/data collection and threat discovery. However, different platforms have strong heterogeneity. SmartThings uses Groovy as its programming language while Google Home uses Kotlin & Java and Homekit uses Swift & Objective C. Besides, the code framework of some platforms are not open source such as Apple Homekit, and provide developers with limited access permission. The traditional code instrumentor is not universal. Furthermore, there are some differences in log formats and traffic patterns between the two different platforms, which increases the difficulty of transplanting a platform-specific framework to other platforms.

In this paper, we introduce CP-IoT, a cross-platform monitoring system for ensuring the automation works properly and discovering the high-risk threats among different automation rules. We develop a universal semantic analyzer that can extract rules and user policies from app pages of different platforms. CP-IoT automatically constructs a dynamic centralized graph to integrate all automations, which portray their normal behaviour and the state transition of devices. Based on the behavior model, CP-IoT provides two detection and defense methods, anomaly detection algorithm for individual automation and cross-rule threats mining method. We propose a multi-granularity clustering approach to acquire the event fingerprint from encrypted traffic of different platforms and identify the cause-effect sequence, which reflects the runtime behavior of an automation rule. We apply the model checking technique to check whether an individual rule violates its specification and detect the inconsistent behavior. To detect cross-rule threats, we formalize each fine-grained type as the path constraints and search all feasible solutions on the graph.

In summary, we make the following contributions:

- We introduce an automation monitoring system CP-IoT[1], which is compatible with different IoT platforms and can detect wide automation threats, including the inconsistent behavior of individual automation rule, cross-rule interactions and interference. We also present the mitigation solutions for each automation threat by analyzing the causes.
- We propose a new cross-platform rule extraction method. By analysing various semantic information (description, configuration) of different app interfaces, it is able to adapt to the architectural differences and interface differences of different platforms.
- We design a novel graph-based behavior model that is able to perceive device state changes under the complex control of automations on different platforms. Moreover, the model is centralized. Combining with a graph-based mining algorithm we design, we can explore various kinds of cross-rule threats more completely, both single-platform and cross-platform.
- We design a multi-granularity fingerprinting approach via side channels, which can identify the runtime behavior of rules on different platforms by extracting packet-level and flow-level fingerprints. Then a graph-based anomaly detection algorithm is designed to discover the rule execution anomalies that are inconsistent with their specifications.
- We validate CP-IoT on four platforms and two testbeds with 1076 automation rules in total. The evaluation result shows that CP-IoT has excellent cross-platform compatibility and is efficient to discover various automation threats.

[1]https://github.com/colinLH/CP-IoT

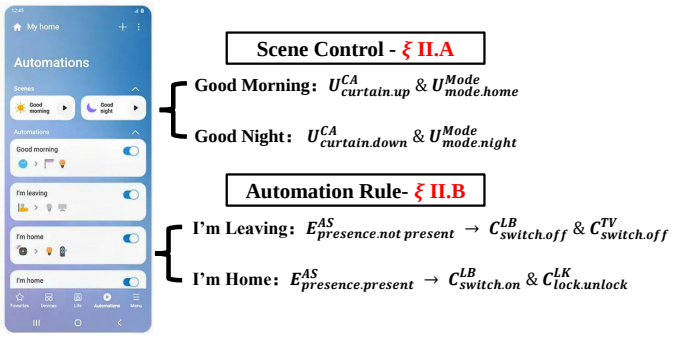


Fig. 1: The symbolic representation of automations.

II. Background: Automations

Mainstream IoT platforms offer official apps to allow users to control their devices remotely and customize their daily routines. In this section, we describe two automation types, scene control and automation rule respectively.

A. Scene Control

After devices are deployed to the smart home, the user can create different scenes such as the living room through the *room/group* option and assign certain devices to them. In the different scenes, users can remotely control the device or modify its properties. Two scene control cases are listed in **Fig.1**, where ***Good Morning*** makes the curtains rise and sets the system mode to home, and ***Good Night*** turns all lights off and sets the system mode to night.

B. Automation Rule

Unlike scene control, the automation rule is a passive control policy. After a rule is deployed to the application, the platform will decide whether to send a controlling command to the specified device based on whether the scene meets the trigger conditions. **Fig.1** shows an automation rule ***I'm Home*** deployed in SmartThings to unlock the door and turn on the light when the user's arrival is detected.

C. Symbolic Representation

Existing researches [21], [22] show that most smart home automations can be represented as IFTTT paradigms. We abstract them as "*IF trigger THEN action*" and convert them into the symbolic representation to facilitate the description of automations. We use the symbol $\boldsymbol{E}$ and $\boldsymbol{C}$ to present the trigger and action part, as the trigger condition can also be referred to an **Event** and the action can be seen as a controlling **Command** sent to devices. Both event and command have specific content and associated devices, which we present with subscripts and superscripts of $\boldsymbol{E}$&$\boldsymbol{C}$ respectively. For example, *"AS"*, *"LB"* and *"TV"* in automation rule ***I'm Leaving*** indicates arrival sensors, light bulbs and television, and *"LK"* in automation rule ***I'm Home*** represents smart lock. Similarly, we use the symbol $\boldsymbol{U}$ to represent the scene control made by the user, and ***Good Morning*** can be expressed as $\boldsymbol{U^{CA}_{curtain.up}}$ & $\boldsymbol{U^{Mode}_{mode.home}}$, where *"CA"* and *"Mode"* denote smart curtain and system mode.

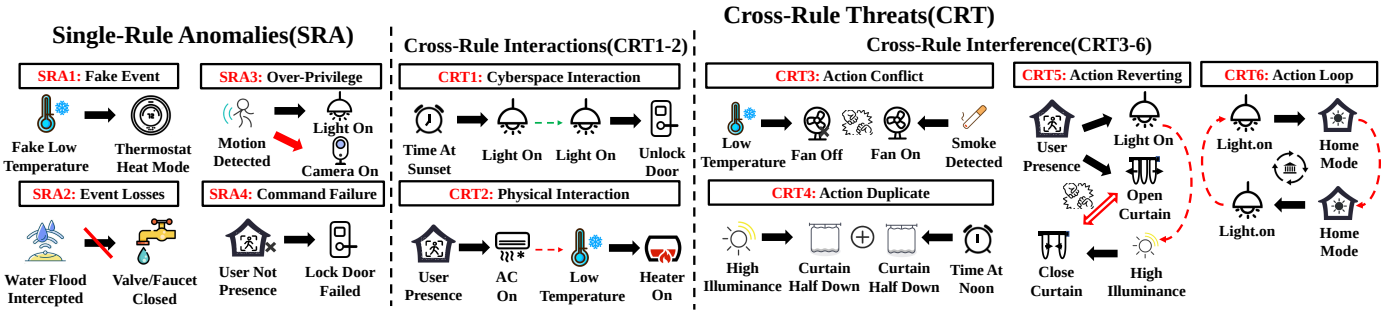


Fig. 2: Examples of various types of threats in the smart home.

TABLE I: Comparison of the state-of-the-art HA defense systems with CP-IoT, where ○ means no support, ◐ means partially support, while ● means totally support.

Method \ Support	SRA	CRT1-2	CRT3-6	Cross-platform
Homonit [14]	◐	○	○	○
HAWatcher [16]	◐	○	○	◐
IoTGaze [15]	●	●	○	◐
Soteria [17]	○	◐	◐	○
HomeGuard [20]	○	●	◐	○
IoTGuard [18]	○	◐	◐	◐
iRuler [19]	○	●	●	◐
CP-IoT	●	●	●	●

III. Motivations, Threat Model and Challenges

In this section, we first discuss the importance of designing an approach that is compatible with different platforms and can detect a wide range of HA system threats. Then we present our threat model including the covered attacker capabilities. Finally, we illustrate the weaknesses of existing methods and describe the challenges to be faced in addressing these issues.

A. Why a cross-platform, full-featured system is needed?

Distributed Device Deployment. Existing HA platforms fail to support all device brands, resulting in devices being deployed on different platforms. Moreover, users prefer to deploy their devices on locally popular HA platforms such as Apple Homekit, Google Home, Amazon Alexa in America and Xiaomi Home in China. Consequently, there is a need to design a HA defense system that supports different platforms, which can meet the varied needs of users in different regions.

Various security threats of automations. All kinds of threats to automations are discussed daily on the communities of various HA platforms [8]–[10]. In a nutshell, the security threats in the application layer of HA systems can be divided into two groups. One is the incorrect behavior of automations, where we call **single-rule anomalies (SRA)**. The other is the potential interaction or interference between two or more automation rules, which we call **cross-rule threats (CRT)**. Unlike SRA, CRT needs to be further examined to determine if it is highly risky or can be exploited by attackers. Some typical cases of each threat type are presented in **Fig.2**, where both SRA and CRT have more fine-grained subclasses and they are described in detail in the **Appendix A**.

B. Why existing HA systems are insufficient?

Low Cross-Platform Compatibility. Most of the existing studies are designed for Samsung SmartThings [14]–[17], [23]–[25], and a few are for IFTTT [26], [27], Google Home [28], which achieve good performance on specific platforms, but not been validated on cross-platform compatibility or unable to support other platforms.

Limited Threat Coverage. Existing researches [14]–[20] have achieved great success in detecting threats of automations. However, they can only cover limited categories in the threat plane. We summarize seven state-of-the-art HA defense methods in **Table.I** and compare their detection capabilities across different threats. Homonit [14] and HAWatcher [16] are two typical systems to detect SRA. However, Homonit can only detect SRA1 and SRA3 while HAWatcher fails to detect SRA3. IoTGaze [15] fails to detect SRA2 but can discover both cross-rule interaction types by vulnerability detection.

The subsequent four methods [17]–[20] aim to detect cross-rule threats(CRT). Soteria [17] fails to detect CRT2 and CRT4, HomeGuard [20] fails to detect CRT2 and CRT6 and IoTGuard [18] fails to detect CRT4. iRuler [19] can detect all CRT. However, none of these methods is able to detect SRA.

In summary, these methods only detect SRA or CRT but not both (except IoTGaze), or cover limited threat subclasses of SRA and CRT. Moreover, these methods are designed for specific platforms (including the working logic and rule extraction methods) or validated in certain platforms, so they have poor compatibility to different platforms.

C. Threat Model

We aim to detect the four SRA types and the six CRT types described in **Fig.2**. CP-IoT can detect SRA caused by device failures, network delays, and attacker behaviours. The SRA detection plane covers four attacker capabilities: **(1) Fake Events**. The attacker constructs and injects malicious packets such as fake user activity events, which result in rules being illegally executed. **(2) Event Interception**. The attacker intercepts some or all of the packets sent by the device or platform (event or command), resulting in partial execution or failure of the rule. **(3) Over-Privilege**. The attacker does not interfere with the execution of the rule, but injects additional command packets to control unrelated devices. **(4) Command Failure**. The attacker intercepts the control commands sent by the platform and disables the "Action" part of the rule. We clarify that CP-IoT can only detect anomalies that violate fingerprint features and time-order characters. Attackers trying to circumvent defenses need to have knowledge of the traffic

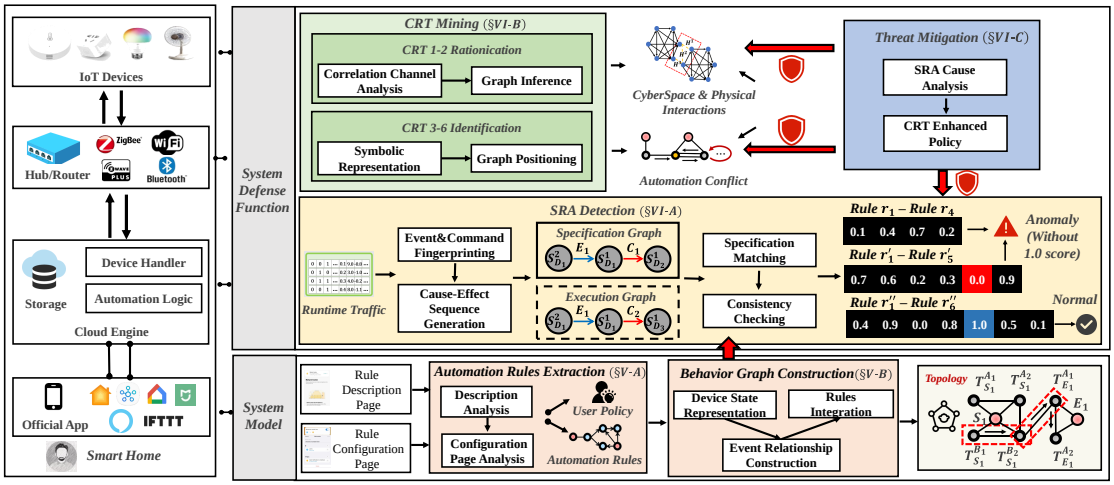


Fig. 3: The two-layer architecture of CP-IoT.

pattern generated by the rule execution as well as the context state of the device at runtime.

Unlike the SRA detection target, CP-IoT aims to find CRT caused by **user misconfiguration**. Specifically, the detection goals are as follow: **(1) CRT1-2**. Users deploy multiple **flawless** rules on the same or different platforms that generate unexpected interactions. **(2) CRT3-6**. Users deploy multiple **flawless** rules in the same scenario, but they interfere with each other, resulting in conflicts, repeated execution of a command or cyclic execution. Here we do not consider the malicious logic injected by the attacker, since most of the rules are defined by the users themselves or recommended by the official platform. Finally, we assume a pristine environment (containing no anomalies) during the training phase of fingerprint extraction and IoT platforms are hard to be comprised.

D. Challenges in Cross-platform Monitoring

Ch1: How to extract automation rules from heterogeneous smart home platforms? A common way is analyzing the source code [14], [15], [24], [29], [30]. However, there is a strong heterogeneity between different IoT platforms that they have various programming languages or the code frameworks are not open source, so the code-based analysis method fails to deal with the architecture difference. Another way uses the rule description of UI pages [15], [31], [32]. However, app interfaces of different platforms have varied description granularity. For example, Homekit has vague descriptions for automations rules. Moreover, most of the descriptions do not contain user policy such as a threshold for temperature to be determined to be high. Even worse, they are designed for platform-specific apps and fail to deal with the interface difference across different platforms.

Our insight is to add the semantic analysis of the configuration pages based on the description analysis, which are shared between different platforms. Moreover, it contains fine-grained rule definitions and user policies, and two phases results can complement each other to achieve high accuracy.

Ch2: How to identify the runtime behavior of rules of different platforms? Before detecting the SRA, we first need to identify the runtime behavior of rules. Some researches find changes in device state through logs [16] or code instrumentation [24], [33] to check the process of rule execution. However, some platforms do not provide access to the log or not open source for security protection. Some other studies [14], [15], [34] analyze traffic to infer the specific events. However, in the multi-platform context, their approach is too coarse-grained to capture the dependencies among events. For example, a device event associates multiple rules from different platforms. When this event is detected, all these rules are triggered. These approaches fail to differentiate the runtime behaviour of rules deployed on different platforms from the complex traffic.

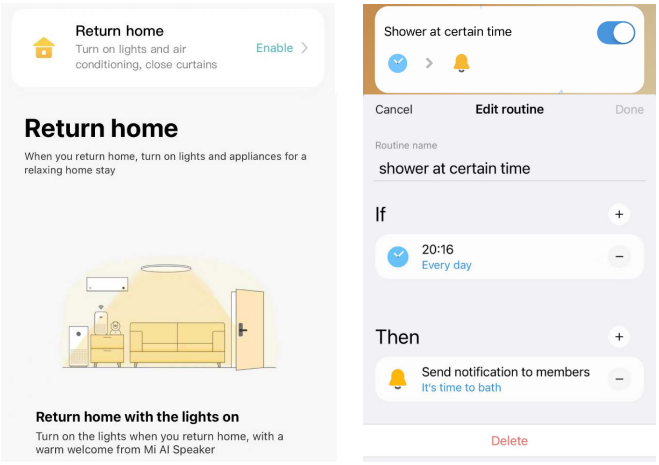
Our insight is to extract both packet-level and flow-level fingerprints of events, where the packet-level fingerprint identifies events and combined with the flow-level fingerprint to distinguish between different platforms. Finally, the events are combined to obtain the runtime behaviour of each rule.

E. Challenges in Defense of Various Threats

Ch3: How to ensure the completeness of detection results? Existing methods construct rule-independent models such as DFA [14], transition graph [15] and logical expressions [16], [17], [19], [23] to portray the rules behavior. However, SRA detection requires both the execution state of the rules and the context state of the devices. They just model the execution point of the automation rules rather than the device state. So they can only be applied to specific platforms, where the device state can be acquired through the logs. Moreover, their model portray each automation rule independently. They fail to capture the complex relationships and dependencies among multiple rules deployed on different platforms such as the action reverting across many platforms.

Our insight is to construct a centralized graph as the behavior model. By modelling each device as different nodes and integrates all automation rules deployed on different platforms, it can track the complex device state changes under the control of multiple rules and provide a more complete exploration space for CRT mining.

Ch4: How to mitigate the risk caused by these threats? Even if existing methods [14]–[16], [35] are able to detect anomalies in various parts of the automation rules with high precision, they fail to provide specific mitigation measures for fine-grained causes. For example, events may be lost for two different reasons: huge network delays or attacker interception. Our insight is to prescribe the right remedy for different SRA



(a) The description page of automation rule "Return Home" in Xiaomi Home.

(b) The configuration page of automation rule "Shower at certain time" in SmartThings.

Fig. 4: The description page and the configuration page.

types and repeat the rule execution to determine whether the anomaly is from a device failure/network cause or an attack.

Furthermore, existing studies rarely consider how to mitigate CRT. When there exists a CRT between multiple automations, removing any one of them could be a strategy, but not feasible, as it does not minimise the impact on users. Our insight is adding a security policy between two more automations or supplementing a few automation rules to block the dangerous cross-rule interactions or interference.

IV. System Overview

In this section, we introduce the CP-IoT workflow and the key modules as shown in **Fig.3**. Before discovering various automation anomalies and threats, we firstly need to obtain specifications of different automation rules. We introduce a universal rule extraction method in section V-A to solve **Ch1**, which only requires the app pages of different platforms. Based on the results, we build a centralized graph model in section V-B to portray the normal behaviour of automations and solve **Ch3**, which is also highly extensible including the addition of new devices and new automation rules of different platforms.

Based on the system model, we design defense functions for detecting SRA and CRT. We introduce the **SRA Detection** module in section VI-A, where we propose a side-channel analysis method for identifying the runtime behavior of automation rules to solve **Ch2**, and apply model checking to detect the specification violation. The **CRT Mining** module runs in parallel with SRA Detection module, which are described in section VI-B. Specifically, we convert each fine-grained CRT subclass into the symbolic representation as path constraints, and search all the rule-pairs or rule-chains on the model that satisfy these conditions.

Finally, the **Threat Mitigation** module (section VI-C) interprets the detection results and generates actionable recommendations to mitigate SRA and CRT to solve **Ch4**. Specially for CRT, we generate enhanced policies to break the interactions and interference, which have minimal impact on the existing rules.

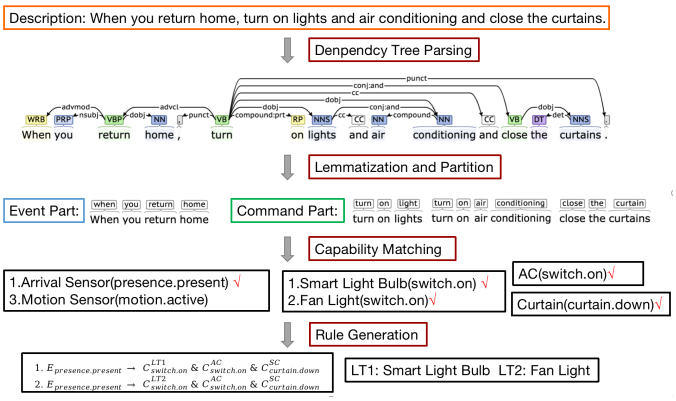


Fig. 5: The workflow of app descriptions analysis.

V. System Model Construction

In this section, we firstly introduce a universal automation rule extraction method for different HA platforms. Then we construct the behavior model of the automations. The following defense functions are developed based on this behavior model.

A. Automation Rules Extraction

Traditional code-based extraction methods [14], [18], [20] can achieve high accuracy but are platform specific, such as designing a code parser in the Groovy language for the SmartThings platform. In addition, some platforms such as Apple Homekit do not support open source. An observation is that most IoT platforms offer official apps that allow users to customize automation. Based on this insight, we propose the following app page based analysis approach.

1) Description Analysis

As shown in **Fig.4(a)**, the description of the automation rule ***"Return Home"*** contains its working logic and we identify the Event part (**E**) and Command part (**C**) by using the NLP tools StanfordCoreNLP [36]. In detail, the workflow of our analysis method is depicted in **Fig.5**. Using the dependency tree parsing, we get the part-of-speech of each word and the dependencies between words. Afterwards, we determine the causality of the sentence according to **Wh-adverb (WRB)** and **punction (punct)**, and divide it into Event Part and Command Part. Considering the complex syntax still exists, we apply lemmatization on both two parts. Specifically, we consider each noun compound and verb compound as a whole, such as "turn" and "on", "air" and "conditioning". Based on the direct object **(dobj)** dependencies between the verb (**VB/VBP**) and the noun (**NN/NNS**), we take pair-wise combination on them to obtain a minimal representation of each event and command. For example, the verb compound **"turn on"** and noun compound **"air conditioning"** are combined as a command clause.

Considering that different devices of the same type have similar attributes, which we denote here by **capability**, we can simplify each clause extracted above. For example, **turning on/off the light** can be abstracted into **"switch.on"/"switch.off"** , where **"switch"** is a capability of the light bulb and **"switch.on"** is a **capability-value** pair.

Consequently, the key part is to match the description clauses with a most similar pair. We combine each capability and its values from the SmartThings Developer Documentation

[37] into the capability-value pair as the ground truth, which provides a good abstraction of device events across different platforms. Then we use the pre-trained BERT [38] model to obtain the embedding of each string and use cosine similarity to compute the correlation between the clause and each capability-value pair. Finally, the capability-value pair with the highest similarity score is used as the matching result.

2) Configuration Page Analysis

Specifically for some capabilities such as temperature and humidity, their values are generally in the NUMBER range. For example, temperature takes the value range of number [-460..10000]. However, the events associated with these capabilities are generally above a certain value (temperature.high), below a certain value (illuminance.low), or reaching a certain value (time.9 am). Generally, the user sets a specific threshold, which we call here the **user policy**. Specifically, we match the contents of the **IF/Condition/Trigger/WHEN** block with the Event Part of the rule and the contents of the **THEN/Action/Adjust** block with the Command Part of the rule. As shown in **Fig.4(b)**, the **"20:16"** in IF block is related to **"time.startTime"**. Similarly, we obtain the embedding of the content in each block by the BERT model and calculate the similarity with each capability-value pair.

It is worth pointing out that the results of this step are also used to complement section V-A1, since the description of some automations are vague and does not contain explicit trigger conditions or actions. For example, the description of the automation rule in **Fig.4(b)** does not contain the action of sending a message to user, while we can know the action **"msg.send"** in the configuration page.

B. Behavior Graph Construction

In this section, we create a centralized graph that integrates various devices and binds various automation rules deployed on different platforms. We make each IoT device as a center instead of building the graph as automation-centric.

1) Device State Representation

Each device may have multiple capabilities, whose values together determine the state of the device at a given moment. For example, a Door Sensor has two capabilities, "contact" and "acceleration". **{"contact": "open", "acceleration": "active"}** determines the state of the Door Sensor, reflecting the detection of opening a door.

Instead of using a single node to represent all the capability values of a device (**scheme1**), we just represent the value of one of its capabilities (**scheme2**). Assume that a device has N capabilities C_i, each with M_i values(average m values), the complexity of scheme1 is $\prod_{i=1}^{N} M_i \sim O(m^N)$, while scheme2 is $\sum_{i=1}^{N} M_i \sim O(mN)$. So the scheme2 can greatly reduce the space complexity of the graph model. Unlike scheme1, the "State" node of scheme2 does not directly reflect the state of the device since only one of the capability values is represented. Therefore, we record all the values of capability in the device node. As shown in the upper left part of **Fig.6**(①), we build a **Device** node for each device and some related **State** nodes to represent the value of its capabilities. Finally, we construct an **Scene_Control** edge between each "Device" node and its related "State" nodes, since scene control made by the user can directly change the state of a device.

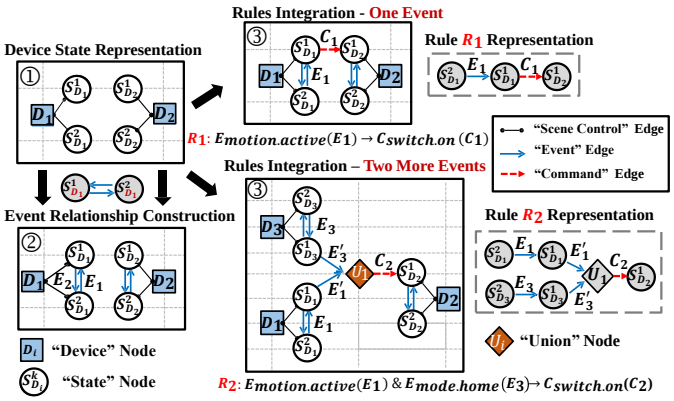


Fig. 6: The workflow of building the centralized graph.

2) Event Relationship Construction

An event occurs when the state of a device changes, which is reflected in the graph as a transition from one "State" node to another. As depicted in the lower left part of **Fig.6**(②), we construct two **Event** edges in opposite directions between two **State** nodes that have the same capability. Besides, each event is recorded in the **Event** edges, which are also represented by the capability-value pair.

3) Rules Integration

The **event part** of the automation rule can simply be located on the graph by matching the "Event" edge attribute with the capability-value pair. At the same time, all the "State" nodes that reached after each event is executed can also be positioned in the graph, which we call **event successor** nodes. In particular, if the rule contains multiple events, this means that they need to occur together before generating the following command part. Therefore, we construct **Union** nodes to represent the intermediate state where all preconditions are satisfied. Two examples are illustrated in the right part of **Fig.6**(③). For the **Rule** R_1, we locate the event successor node $S^1_{D_1}$ after the event E_1 is matched. For the **Rule** R_2, we create "Union" node U_1 and the "Event" edges (E'_1, E'_3) between U_1 and each event successor nodes $(S^1_{D_1}, S^1_{D_3})$.

When the command is executed, the relevant capability of the device will be set to a specific value. For example, **"switch.on"** will set the light's property to **{"switch": "on"}**. Based on this, we can find all the "State" nodes ($S^1_{D_2}$ of Rule R_1, R_2) that reached after each command is executed, which we call **command successor** nodes. If the rule contains multiple events (Rule R_2), we build the **Command** edges (C_2) between the "Union" node and each command successor node. Otherwise (Rule R_1), we construct **Command** edges (C_1) between the event successor node and each command successor node. Finally, we assign an **ID** to each rule, recorded by all nodes and edges associated with the rule.

VI. System Defense Function Design

In this section, on the basis of system model, we propose two automation defense methods that cover various types of security problems in SRA, CRT. Finally, for each threat type,

TABLE II: Five statistics used for fingerprint extraction.

Type	Statistic	Notation	Description
Packet	Size	s_1	Packet size will vary from event to event
	Protocol	s_2	WiFi(0), Z-Wave(1) Zigbee(2), Bluetooth(3)
	Direction	s_3	0: $device \rightarrow router$ 1: $router \rightarrow device$
Flow	Interval	f_1	Average packet interval
	Length	f_2	The length of packet sequence

we propose corresponding mitigation methods based on the detection results, which enhance the security of smart home.

A. SRA Detection

Since some platforms give limited permission to developers, it is impossible to determine whether the automation is performing correctly just by the information of device state. For this reason, we firstly identify the cause-effect sequence generated by the rule execution from the real traffic, and then perform a consistency checking with the specification contained in the system model to judge whether anomalies occur.

1) Motivating Example

To better understand our approach, we use three devices Aqara Motion Sensor (MS), Mi Desk Lamp (L1), Philipus Bedroom Lamp (L2) and two rules $E^{MS}_{motion.active} \rightarrow C^{L_1}_{switch.on}$ and $E^{MS}_{motion.active} \rightarrow C^{L_2}_{switch.on}$ as an example. These two rules are executed when $E_{motion.active}$ is triggered. Suppose the generated traffic is $\{P_1, P_2, ..., P_8\}$, where packets P_1, P_2 are generated by $E_{motion.active}$, P_3, P_6, P_7 by $C^{L_1}_{switch.on}$ and P_4, P_8 by $C^{L_2}_{switch.on}$. As shown in **Fig.7**, each flow (flows1-3) corresponds to a device behavior and we can know the specific content by matching it with event/command fingerprints. The Cause and Effect parts can be divided easily based on the interval. By combining each matched part, we can get the possible runtime behavior of the rules. Finally, we convert these sequences into execution graphs and use them as input to anomaly detection algorithm with the behavior model. If an exact match of the specification is found, the rule is determined to be executed correctly.

2) Event&Command Fingerprinting

When an automation rule is triggered, there is traffic between the associated device and the router. We firstly filter the noisy and unrelated traffic, including beacon packets and retransmission packets. Since events and commands affect different devices independently, we extract data flow between each device and router. Assuming that the filtered packet sequence is $P = \{p_1, p_2, ..., p_N\}$ and the related device number is $\boldsymbol{Q}$, the sequence can be separated into $\boldsymbol{Q}$ parts:

$$P = \{P_1, P_2, ..., P_Q\} \quad s.t. \sum_{i=1}^{Q} |P_i| = N \tag{1}$$

where **each sub-sequence P_i represents the traffic flow generated by the execution of an event E_i or command C_i.** Next we construct the fingerprint of each event $\boldsymbol{E_i}$ or command $\boldsymbol{C_i}$ by extracting packet-level features and flow-level features

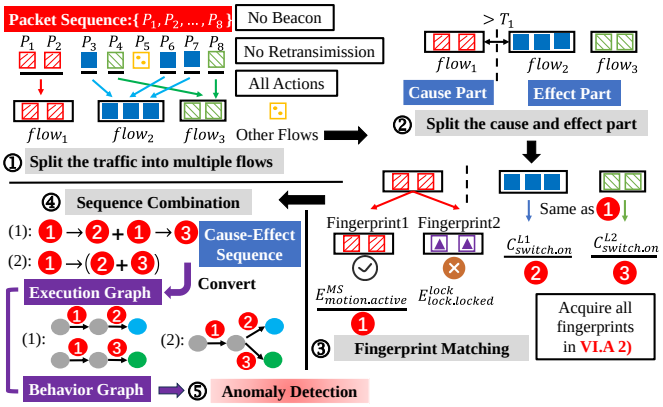


Fig. 7: A motivating example for identifying the runtime behavior of the automation rule.

as described in **Table.II**. Suppose each flow P_i has $\boldsymbol{s}$ packets, the flow-level fingerprint is denoted a 2-tuple $\boldsymbol{f_{P_i}}$:($\boldsymbol{f_{i,1}}$, $\boldsymbol{f_{i,2}}$), and the packet-level fingerprint $\boldsymbol{m_{P_i}}$ can be represented as the following feature matrix:

$$m_{P_i} = \begin{Bmatrix} p_{1,s_1} & p_{1,s_2} & p_{1,s_3} \\ p_{2,s_1} & p_{2,s_2} & p_{2,s_3} \\ \vdots & \vdots & \vdots \\ p_{s,s_1} & p_{s,s_2} & p_{s,s_3} \end{Bmatrix} \tag{2}$$

We trigger $\boldsymbol{T}$ times for each automation rule. For each flow, we get a flow-level fingerprint and a packet-level fingerprint each time. We use K-Means [39] (cluster=1) to cluster all T times generated fingerprints to eliminate the deviation and use the clustering centers as the flow-level fingerprint $\boldsymbol{F_i}$ and packet-level fingerprint $\boldsymbol{M_i}$ of event E_i/command C_i.

3) Cause-Effect Sequence Generation

After extracting the event/command fingerprints, we can match them in the runtime traffic. Specifically, we still split the traffic into device-router flows, and the start timestamp of each flow reflects the sequence of actions, namely the cause and effect relationship. Suppose the flow with the earliest timestamp $\boldsymbol{t_1}$, we classify all flows with time interval up to $\boldsymbol{T_1}$ from $\boldsymbol{t_1}$ as the **Cause** part, while the remaining are been seen as the **Effect** part ($\boldsymbol{T_1}$ is a predefined parameter obtained through a large number of experimental attempts). Then we match the runtime traffic of each part with the fingerprints.

$$\underset{E_i/C_i}{argmin} \left(\underbrace{\lambda \cdot D\left(F_i,\ f_j\right)}_{Flow\ d_f} + \underbrace{\delta \cdot D\left(M_i, m_j\right)}_{Packets\ d_p} \right) \quad s.t.\ d_f + d_p \leq d \tag{3}$$

As illustrated in **Equation.3**, assuming that the flow feature of the j^{th} incoming flow is $\boldsymbol{f_j}$ and the packet feature matrix is $\boldsymbol{m_j}$, we use the Manhattan Distance as the distance metric function ($\boldsymbol{D}$) to calculate their similarity with the fingerprint of event E_i/command C_i ($\boldsymbol{F_i}$, $\boldsymbol{M_i}$). Considering that flow-level features are more coarse-grained than packet-level features, we prefer $\boldsymbol{\lambda < \delta}$. Then we take the event or command as the matching result whose fingerprint has the minimum distance with the features of the j^{th} flow. However, if the weighted sum of flow distance d_f and packet distance d_p is greater than the threshold $\boldsymbol{d}$, we consider that this flow does not match with

any event or command. Finally, we can generate the cause-effect sequence such as $\boldsymbol{E_{motion.active} \rightarrow C_{switch.on}}$ by matching the **Cause** part and **Effect** part with all fingerprints.

4) Anomaly Detection

After identifying the cause-effect sequence in the open environment, we judge whether SRA occur by checking its consistency with the rule specification. We divide the process of SRA detection into two stages: **specification matching** and **consistency checking**, which are described in **Algorithm 1** in **Appendix B.A**.

Specification Matching. This step is used to identify the rule specification on the graph that is most similar to the cause-effect sequence. We first split the cause-effect sequence into cause set $\boldsymbol{S_c}$ and effect set $\boldsymbol{S_e}$ with size $\boldsymbol{l_1}$ and $\boldsymbol{l_2}$ (line 1). For each element in both sets, we find the rule on the graph that contains that event/command, returning the corresponding rule id $\mathcal{P}_{id}$. Then we find the path between each begin "State" node of "Event" edges and each end "State" node of "Command" edges, and combine all paths as the rule **specification graph** $\mathcal{G}_{sp}$. Two examples of the generated rule specification graph are depicted in the dashed box in **Fig.6**.

Consistency Checking. According to the method discussed in section V-B3, the cause-effect sequence is transformed to a graph, which we call the rule **execution graph** $\mathcal{G}_{cs}$. Above all, SRA2 need to be detected separately since the device state information from the platform logs is required. When the platform logs records a state change in certain device while two sets ($\boldsymbol{S_c}$, $\boldsymbol{S_e}$) of the cause-effect sequence are empty, it means that the device makes an action but the generated events are lost or intercepted and a SRA2 case occurs (line 2-3).

Then we match the execution graph $\mathcal{G}_{cs}$ with each specification graph $\mathcal{G}_{sp}$ to determine whether the execution of the rule deviates from the behavior profile (line 4-6). Specifically, we use GraphSAGE [40] to learn the graph representations of $\mathcal{G}_{cs}$ and $\mathcal{G}_{sp}$, including topology, node features and edge features. Finally, we use cosine similarity to compute the similarity $\boldsymbol{s_{total}}$ between two embeddings, and the similarity of event part $\boldsymbol{s_e}$ and command part $\boldsymbol{s_c}$ (line 7-11). When the matching process is completed, a similarity list $\boldsymbol{S_l}$ is obtained. Finally, the algorithm detects an anomaly occurs when not exists an exact matching ($\forall s \; in \; S_l, \; s[0] \neq 1$)(line 12-15). The specific SRA types will be identified in Section VI-C.

B. CRT Mining

In this section, we firstly ratiocinate the interaction threats between two automation rules (CRT1-2). Based on this, the cross-rule interference are located on the graph (CRT3-6).

1) Cross-Rule Interactions Ratiocination

For a better description, we symbolize the flow of the cross-rule interaction as $(\boldsymbol{R_1} : \boldsymbol{E_{r1}} \rightarrow \boldsymbol{C_{r1}}) \rightarrow (\boldsymbol{R_2} : \boldsymbol{E_{r2}} \rightarrow \boldsymbol{C_{r2}})$. For the cyberspace interactions, the command of rule R_1 directly triggers the rule R_2, so the command set C_{r1} of rule R_1 must contain the event set E_{r2} of rule R_2, namely the constraint $C_{r1} \supseteq E_{r2}$. For the physical interactions, some commands in the command set C_{r1} of rule R_1 change the physical environment and indirectly triggers rule R_2. For example, turning on the light ($E^{light}_{switch.on}$) increases light intensity ($E^{sensor}_{illuminance.high}$). To discover such interactions, we firstly identify the correlation between each event/command through the physical channel.

TABLE III: The symbolic representations of four cross-rule conflict types. Specifically, $\boldsymbol{S_i}$ refers to a "State" node, $\boldsymbol{C_{ri}}$ refers to the command set of rule $\boldsymbol{r_i}$, $\boldsymbol{E_{ri}}$ refers to the event set, $\boldsymbol{S_i^s}$ and $\boldsymbol{S_i^p}$ refers to the successor node and predecessor node of a "Event"/"Command" edge, $\boldsymbol{cp}$ and $\boldsymbol{val}$ refers to the "capability" and "value" attributes of "State" node $\boldsymbol{S_i}$.

Type	Representation
Action Conflict CRT3	$\exists q_1 \in C_{r1}, q_2 \in C_{r2}$ $S_1^s = q_1.suc, S_2^s = q_2.suc$ $s.t. \;\; S_1^s.cp = S_2^s.cp, S_1^s.val \neq S_2^s.val$
Action Duplicate CRT4	$\exists q_1 \in C_{r1}, q_2 \in C_{r2}$ $S_1^s = q_1.suc, S_2^s = q_2.suc$ $s.t. \;\; S_1^s.cp = S_2^s.cp, S_1^s.val = S_2^s.val$
Action Reverting CRT5	$\exists q_1 \in C_{r1}, q_n \in C_{rn}$ $S_1^s = q_1.suc, S_n^s = q_n.suc$ $s.t. \;\; S_1^s.cp = S_2^s.cp, S_1^s.val \neq S_n^s.val,$ $\forall_{i=1}^{n-1}(r_i, r_{i+1}) \in S_{cyb}/S_{phy}$
Action Loop CRT6	$s.t. \;\; E_{r1} \subseteq C_{rn}$ $\forall_{i=1}^{n-1}(r_i, r_{i+1}) \in S_{cyb}/S_{phy}$

Correlation Channel Analysis. We consider **11** physical channels, including *airQuality*, *acceleration*, *carbonDioxide*, *carbonMonoxide*, *illuminance*, *humidity*, *motion*, *smoke*, *sound*, *temperature* and *water*. After that, we analyze the correlation between **22** capability-value pairs associated with these 11 channels and the remaining **341** capability-value pairs extracted from the SmartThings Document in section V-A1. Similarly, we use the BERT model to calculate the similarity scores and record all related channels with score above $\boldsymbol{T_2}$ ($\boldsymbol{T_2}$ is a predefined parameter to select the strongly associated capability-value pairs).

Graph Inference. The graph inference algorithm is illustrated in **Appendix B.B**. We firstly find all the rules and combine each two of them as a pair. To find the physical interactions, we convert each command into a capability-value pair according to the most correlated physical channel, such as transforming $C^{fan}_{switch.on}$ to $E^{sensor}_{temperature.low}$. Finally, if two rules r_1, r_2 satisfy the constraints($C_{r1} \supseteq E_{r2}$), they are added to the result set $\boldsymbol{S_{cyb}}$ or $\boldsymbol{S_{phy}}$. For the subsequent identification of CRT3-6, we construct **Physical** or **Cyber** edges between the command successor "State" node of each $\boldsymbol{R_1}$ and the event predecessor "State" node of each $\boldsymbol{R_2}$.

2) Cross-Rule Interference Identification

After mining the cross-rule interactions (CRT1-2), the behaviour profile becomes more complete and have included the cross-rule interference. So we only need to locate each threat type on the graph according to its semantic.

Symbolic Representation. The graph representation of each interference type is described in **Table.III**. Among them, CRT3-4 involves two automation rules. We determine whether the commands of two rules control the same device by checking whether the attribute "capability" of "State" node is the same, and determine whether CRT3 or CRT4 occurs by checking whether the attribute "value" of "State" node is different ($S_1^s.val \neq S_2^s.val$) or the same ($S_1^s.val = S_2^s.val$). Different from CRT3-4, CRT5-6 involves multiple automation

rules and a sequence of consecutive triggers for these rules ($\forall_{i=1}^{n-1}(r_i, r_{i+1}) \in S_{cyb}/S_{phy}$). CRT5 have the similar type with CRT4, where the command of two rules conflict. For CRT6, the last rule triggers the first rule again ($E_{r1} \subseteq C_{rn}$), causing a continuous execution of these rules.

Graph Positioning. We convert these representations into constraints by finding any rule group on the graph that satisfy them. For CRT3-4, we return two matching rules (r_1, r_2) and the conflicting commands (c_{r1}, c_{r2}) or the common command part (c_{com}). For CRT5-6, we return the matching rule chain $(r_1, r_2, ..., r_n)$.

C. Threat Mitigation

Aforementioned threats pose significant security risks to the smart home. In this section, we firstly analyze the causes of each SRA type based on the detection result. Then we develop the enhanced security policy to eliminate CRT.

1) SRA Cause Analysis

The interpretation of anomaly detection algorithms can bridge the semantic gap between detection results and executable recommendations. Specifically, we analyze the matching results for each part and determine the type and cause of the SRA. **Fig.8** shows the rule specification and the corresponding execution graph when each anomaly occurs. SRA2 can be detected directly since its events and commands are empty, but the device state transition is recorded. SRA1 occurs when the event part does not match and commands part is matched ($s_e \neq 1, s_c = 1$). For SRA3, Command Interception occurs when the command part does not match at all ($s_e = 1, s_c = 0$), and Command Failure occurs when both the event and command parts match, but the node attributes do not ($S^1_{D_2}, S^2_{D_2} mismatched$). Similarly, SRA4 occurs in two situations, one for controlling a device with unrelated capability and the other for controlling an unrelated device, both of which result in a mismatch in the command part ($s_e = 1, s_c \neq 1$).

Then we analyze the cause for each SRA type and design the corresponding solutions for **non-attacker** device behavior, mainly passive configuration modifications for users. To identify whether the anomaly is made by the attacker, we implement the solution in **Fig.8** and execute the rule again. If the rule is not executed properly, it is judged as the **attacker** behavior (such as interception, injection). Then we will take an active response strategy instead, namely block the execution of the rule, restart the device and send a message to the user.

2) CRT Enhanced Policy

The root cause of cross-rule interactions (CRT1-2) is the user's unawareness of the accidentally triggered rules and the unexpected device behavior. To solve this problem, we **send a message** about all triggered rules to the user and attach the risk level (High/Low) by determining if the command sent by the accidentally triggered rule contains the dangerous actions such as $C_{heater.on}, C_{camera.on}$ and $C_{lock.unlocked}$.

Specifically, we discuss the five security issues and the associated commands in **Table.VIII** of **Appendix B.C**. To address fire and chilly problems, the device will be shut down when the temperature is above or below a threshold. For the water flooding issue, the related device will be switched off

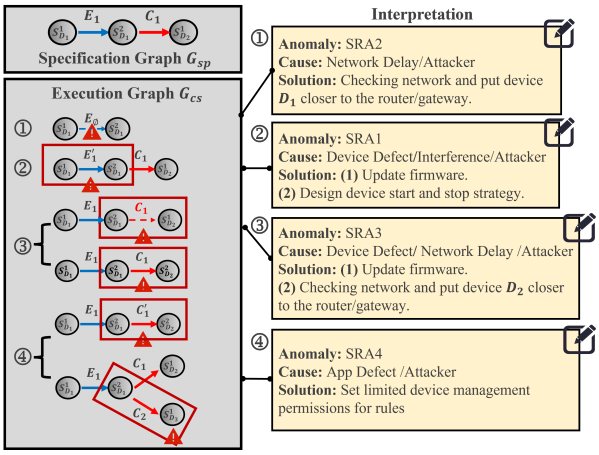


Fig. 8: The interpretation of SRA detection results.

when the sensor detects water. For burglary issues, when the sensor detects the user's presence, the opposite command is executed such as $C_{lock.locked}$. For the privacy leakage problem, the rule turns off the camera when the user is at home.

Each rule may be considered as feasible when they are deployed independently, but they can cause interference or redundancy when combined together. To mitigate the harm caused by the cross-rule interference, we design **enhanced security policies** for each CRT3-6 cases, which also have minimum impact on the existing rules.

1) **Action Conflict & Action Duplicate (CRT3-4)**: we set the priority for each rule and block the execution of lower-priority rule. Specifically, the priority order is User Activity > Physical-Related > Physical-Unrelated > Mode-related > Time-related, since events involving physical channels may cause security problems if they are not handled in time. The Mode-related and Time-related automation rules have the lowest priority since they tend to be routine or periodic.

2) **Action Reverting (CRT5)**: Suppose the first rule is $E_{r1} \rightarrow C_{r1}$ and the final automation rule is $E_{rn} \rightarrow C_{rn}$. To eliminate such threats, we add the condition of contextual state at the last rule, namely we execute this rule only if E_1 does not occur within the nearest time $\boldsymbol{T_{pre}}$. So the final automation rule r_n is modified to E_{rn} & $Happen(E_1) > T_{pre} \rightarrow C_{rn}$. The time threshold T_{pre} varies according to the different rules.

3) **Action Loop (CRT6)**: Suppose the chain of rules that cause the loop is $(r_1, r_2, ..., r_n)$. Since the loop occurs either r_1 or r_n is executed, we modify r_1 as E_{r1} & $Count(r_n) = 0 \rightarrow C_{r1}$ and r_n as E_{rn} & $Count(r_1) = 0 \rightarrow C_{rn}$, where $Count$ records the number of times r_1 and r_n are executed.

VII. Evaluation

In this section, we firstly introduce our experimental setup including a real testbed and a simulation testbed. Then we evaluate the accuracy of the rule extraction method and collect traffic for rule execution to validate the performance of the SRA detection on four mainstream IoT platforms. Finally, we validate the effectiveness of CRT mining on both real testbed and simulation testbed. We also reveal the security implications of discovered threats in **Appendix D.C**. More experimental details are shown in our open source project.

TABLE IV: The accuracy of rule extraction on four platforms.

Platform	Word2Vec [47]		BERT [38]	
	DAnalysis	+CAnalysis	DAnalysis	+CAnalysis
SmartThings(105)	89.52%	92.38%	91.43%	97.14%
Apple Homekit(128)	89.06%	96.09%	92.97%	99.22%
Google Home(160)	83.13%	95.63%	91.25%	98.13%
Xiaomi Home(192)	85.94%	91.15%	88.02%	98.96%

TABLE V: Two cases of Danalysis matched vaguely and Canalysis matched exactly.

Clause	Danalysis Matched	Similarity	Canlysis Keywords
change the light color	$C^{light}_{color.red}$	0.832324	red, color temperature: 1000-1500K
	$C^{light}_{color.yellow}$	0.825504	yellow, color temperature: 3200-4500K
	$C^{light}_{color.blue}$	0.829054	blue, color temperature: 7000-10000K
enter the area	$E^{sensor}_{motion.active}$	0.790236	motion, user activity, action
	$E^{sensor}_{presence.on}$	0.783105	presence, at home, arrival
	$E^{sensor}_{acceleration.active}$	0.767295	acceleration, door/window, contact

A. Experiment Setup

Existing public datasets [41]–[43] do not contain the traffic of home automation collected from different platforms. Moreover, many old devices are no longer compatible with the current app. As a result, we use the following testbed:

Real Testbed. We deploy **32** IoT devices in our lab on four smart home platforms including SmartThings, Google Home, Apple Homekit and Xiaomi Home, The detailed device information and the testbed layout are described in **Appendix C.A**. Since most devices are not compatible with all four platforms, we customize a different number of automations, where **105** rules on SmartThings, **128** rules on Apple Homekit, **160** rules on Google Home, and **192** rules on Xiaomi Home. Moreover, We evaluate the time overhead of adding IoT devices in **Appendix D.A.**

Simulation Testbed. Considering the limited number of automations and device types deployed in the real testbed, we design a simulation testbed which has 7 typical scenes and 54 devices. The device deployment layout of this testbed and the detailed device information is described in **Appendix C.B**. We crawl 10796 applets from the IFTTT [44] website and 82 SmartApp from the SmartThings Public GitHub Repository [45], and filter **2491** rules associated with these devices.

Implementation Configuration. The code is running on Ubuntu 18.04, 128GB RAM, 8vCPU, 1*NVIDIA A100 GPU server, where GraphSAGE for anomaly detection is running in Docker. We use py2neo to build the behaviour model and store it in the Neo4j [46] database, and use NetworkX to convert the data into input for GraphSAGE. For fingerprint extraction, we use a Texas instruments CC2531 USB Dongle and set TiwsPc at channel 15(0x0F) to capture ZigBee traffic, Zniffer to capture ZWave traffic, and EDU Wireless Adapter to collect WiFi and Nordic nRF52840 to collect Bluetooth traffic.

B. Automation Extraction Performance

By checking the consistency of the extracted rules with the event part and command part of the deployed rules (the ground truth), we compare the accuracy of CP-IoT two-stage automation rule extraction using Word2Vec [47] or BERT [38] to acquire the clause embedding, where description page

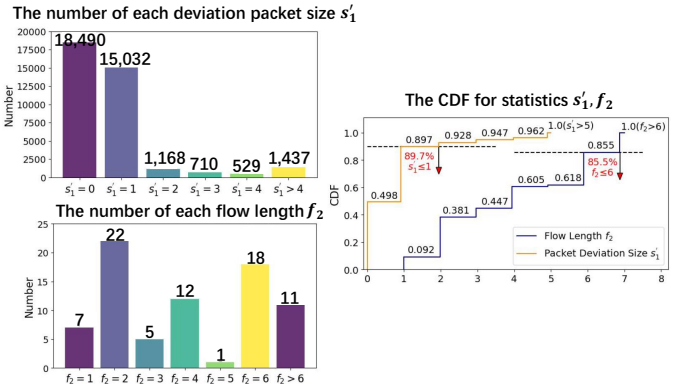


Fig. 9: The distribution of size deviation and flow length.

analysis is used in the first stage and configuration page analysis is added in the second stage.

Model Training. We first train the models using the Google News Dataset [48] containing 100 billion words. After that, we crawl 10135 automation descriptions (sentence) from IFTTT website as domain knowledge dataset to fine-tune and enhance the applicability of the model to smart homes. Finally, we sum and average the embedding of each word in the clause obtained by Word2Vec and return a (300 $\times$ 1) clause embedding. The reason for not using TF-IDF or SIF here is that we consider each word in the clause equally important. In contrast, BERT returns a clause embedding with a larger dimension, i.e., a (1280 $\times$ 1) vector.

Extraction Accuracy Across-Platform. The granularity of rule descriptions varies in different platforms, which affects the accuracy of the first stage. As shown in **Table.IV**, the accuracy of Word2Vec and BERT in the first stage reaches between 80.0% and 93.0%, where **Danalysis** refers to Description Analysis and **Canalysis** refers to Configuration Analysis. We analyze two possible reasons for the matching failure: 1) The description is abstracted. For example, a rule description is "House Freeze Notification" which does not contain Wh-adverb so that model fails to distinguish the Event Part and Command Part. 2) The model matches the clause to the wrong capability-value pair. Two examples are described in **Table.V**, where the command clause "change the light color" has high similarity scores with three capability-value pairs, but matches the capability-value pair $C^{light}_{color.blue}$ every time. In addition, "enter the area" is also strongly correlated with multiple sensor events, but always matches $E^{sensor}_{motion.active}$.

Then the second-stage extraction method CAnalysis greatly increase the identification accuracy improved from 2.86% to 12.50% on Word2Vec and from 5.71% to 10.94% on BERT. We analyze two reasons for this: 1) The detailed partition between triggers and actions in the configuration page. 2) The fine-grained description of each part makes the semantic matching more accurate. Overall, the BERT outperforms Word2Vec with an accuracy improvement of 2.50% to 7.81% and an average accuracy of 98.96% across four platforms, which is capable of cross-platform rule extraction tasks.

C. SRA Detection Performance

We firstly introduce some predefined parameters in event fingerprinting settings. Then we compare the SRA detection performance of CP-IoT with two state-of-the-art detectors on

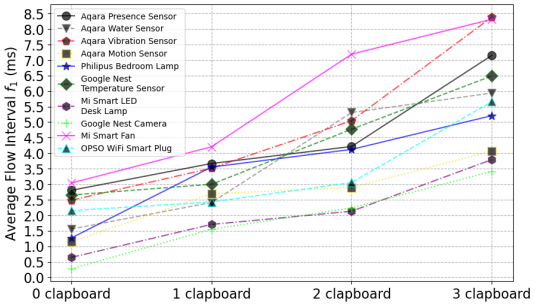


Fig. 10: The flow interval fluctuation.

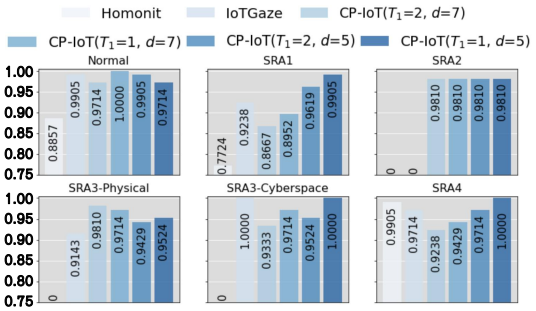


Fig. 11: SRA detection accuracy on SmartThings.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	0x461e	0x0000	ZigBee	113	Data, Dst: 0x0000, Src: 0x461e
14	63.569007	0x461e	0x0000	ZigBee	113	Data, Dst: 0x0000, Src: 0x461e
31	119.243167	0x461e	0x0000	ZigBee	115	Data, Dst: 0x0000, Src: 0x461e
33	126.440160	0x461e	0x0000	ZigBee	113	Data, Dst: 0x0000, Src: 0x461e

Fig. 12: An false negative case of the Aqara Vibration Sensor fake event $E_{vibration.active}$.

SmartThings and validate the performance of CP-IoT on three other platforms.

Dataset. We use the real testbed described in Section VII-A to generate runtime traffic of each automation rule and use it as the normal dataset. Then we use the method introduced in **Appendix C.C** to manually inject generated anomaly cases for each SRA type. Specifically, we inject WiFi packets by aircrack-ng [49], Zigbee packets by KillerBee [50], Z-Wave packets by Z-Attack [51] and Bluetooth packets by BtleJack [52]. For each SRA type, we generate the same cases number as the deployed rule in different platforms.

Evaluation Metrics. We use the following metrics to evaluate the performance of SRA detection. **TP/FN:** Predicted SRA type and the type of injected SRA case is same/different. **TN/FP:** The test case is normal and the predicted type is normal/wrongly predicted as a certain SRA type. **Accuracy:** $\frac{TP+TN}{TP+FP+FN+TN}$, **Precision:** $\frac{TP}{TP+FP}$, **Recall:** $\frac{TP}{TP+FN}$, **F1-Score:** $2 \cdot \frac{Precision \cdot Recall}{Precision+Recall}$, **Nodes:** the node count in the behaviour model, **Edges:** the edge count in the model, **Time:** the running time in seconds.

Baseline. We compare the performance of SRA with two state-of-the-art detectors, Homonit [14] and IoTGaze [15]. Specifically, we build two behaviour models DFA and transition graph for Homonit and IoTGaze respectively. To identify the runtime behavior of the rule, Homonit use Levenshtein Distance [53] to calculate the similarity between the runtime traffic and the fingerprint, while IoTGaze trains a supervised Random Forest (RF) model to measure the similarity. We apply two statistics (s_3, f_2) for Homonit to represent the event & command fingerprint and four statistics (s_1, s_2, s_3, f_1) for IoTGaze as discussed in their paper. Then we trigger each event and command 50 times. For Homonit, we use the Levenshtein Distance to calculate a representative feature among the 50 event features as the event fingerprint. For IoTGaze, we feed all data to RF for training and use the Grid Search with Cross Validation (GridSearchCV) to optimize its hyperparameters.

Traffic Fingerprinting Setup. To filter the traffic flow between router and device or hub and device, WiFi uses IP-address, Zigbee uses MAC address, Z-Wave uses node IDs and for Bluetooth, we connect the phone to the device Bluetooth and capture the flow in adbshell by hcidump [54] tool. As discussed in section VI-A3, we define a time threshold T_1 to distinguish between events and commands. Considering that a lower time threshold may wrongly split the packets of same event into two parts, which mismatch with the fingerprint. A higher time threshold may classify the command part of the traffic as the event part. Through extensive experiments, we try two configurations $T_1 = 1s$ and $T_1 = 2s$.

Moreover, we also define a distance threshold d to distinguish whether a runtime flow belongs to a known or unknown event. We set $\lambda = 0.3$ and $\delta = 0.7$. Since no deviation is allowed between the features of runtime traffic and fingerprint on statistics s_2, s_3, f_2, the tolerance of deviation is given to statistics s_1, f_1.

To obtain the total deviation of s_1, as shown in **Fig.9**, we count the distribution of the packet size deviation $\boldsymbol{s_1^{'}}$ and flow length $\boldsymbol{f_2}$ statistics on 75 event flows associated with 32 devices deployed on the real testbed and a total of 37,366 packets, where the size deviation of 89.7% packets is no more than 1 and 85.5% flow length is less than 7. So we set $s_1^{'} = 1$ and $f_2^{max} = 5$ *or* 6. We calculate the total packet size tolerance deviation of each flow $\boldsymbol{s_1^{total}}$ by the formula $s_1^{total} = s_1^{'} * f_2^{max}$ and get two configs $s_1^{total} = 5$ or $s_1^{total} = 6$.

Due to physical interference, there are network fluctuations and delays which will affect the the flow interval f_1. We measure the fluctuation range of the flow interval on ten representative devices, by placing 0-3 clapboards between each device and the router. The results are shown in **Fig.10** and the flow intervals all within 10ms, so we set $f_1^{'} = 10$. Finally, we combine s_1^{total} and $f_1^{'}$ to calculate the tolerance deviation distance d by the formula $d = \lambda * f_1^{'} + \delta * s_1^{total}$, and adopt two configurations $d = 5$ and $d = 7$. The effect of variations in the parameters T_1, d on the SRA detection performance is discussed in **Appendix D.B**.

Comparison in SmartThings. We first compare the performance of CP-IoT with two baselines on the SmartThings platform since Homonit and IoTGaze does not support other three platforms. ① As shown in **Fig.11**, CP-IoT outperforms Homonit and IoTGaze for both normal cases and each SRA type, with the accuracy of 98.10% or higher in the best configuration. ② CP-IoT($T_1 = 2s, d = 7$) has the highest classification accuracy on normal cases because the relax fingerprint restrictions increase the tolerance of network latency and physical interference, thus reducing the false positive rate($FPR \downarrow$). ③ CP-IoT($T_1 = 1s, d = 5$) has the best performance on detecting SRA1, SRA3-Cyberspace and SRA4, with the accuracy close to 100.0%. To identify these SRA

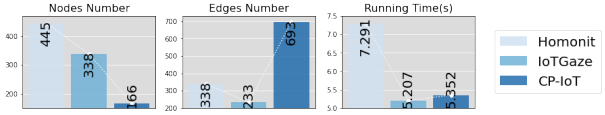


Fig. 13: Comparison of the three behaviour models in terms of space complexity and time complexity.

types, the model needs to strictly determine which part (Event/Command) the traffic belongs to and the difference between fake event/command traffic and normal traffic. Any deviation will classify anomalies as incorrect SRA types and increases the false negative rate($FNR \uparrow$). For example, IoTGaze fails to identify the fake event $E_{vibration.active}$ of Aqara Vibration Sensor, which contains a 2 flow length deviation described in **Fig.12**, and the fake command $C_{screen.shot}$ of the Google Nest Camera, which contains a 1 flow length deviation. ④ SRA2 and SRA3-Physical need to record the device state, while CP-IoT fails to get the logs of Fibaro Smart Button from SmartThings. ⑤ There is no difference in detection effectiveness between the four configurations of CP-IoT on SRA2, since the rule is not executed for the events generated by the device which are not received by the platform. While two baselines do not have the ability to detect this type. ⑥ **Fig.13** compares the runtime cost. CP-IoT nodes number is much less than the baselines since the "State" nodes are shared between different rules, which greatly reduces the redundancy. Although CP-IoT edges number is 2-3 times more than the baselines, CP-IoT provides more functions including the device state tracking and scene control monitoring. The average time of CP-IoT to classify each case is close to IoTGaze only increased by 2.71%, and 26.59% less compared to Homonit.

Detection Result Across-Platform. The detection results of CP-IoT on different platforms are shown in **Fig.14**. It can be seen that the detection precision of different platforms is higher than 99.0%, and the recall of CP-IoT with the best configuration is higher than 98.0%. So the SRA detection method of CP-IoT has good cross-platform compatibility.

Besides, we analyze the detection results and obtain the following two observations: ① There are two reasons for the low CP-IoT recall in some configurations on SmartThings and Xiaomi Home: 1) Fake Events (SRA1) of some small sensors are difficult to be identified since they have small flow length and packet deviation, while Over-Privilege (SRA4) can be easily identified since the execution graph and specification graph have significant differences. 2) Some platforms do not provide log information for certain devices. For Event Losses (SRA2) and Command Failure (SRA3), CP-IoT fails to automatically acquire whether the device state is changed and classify them as normal. ② The transmission rate varies widely across platforms, where some devices of SmartThings and Google Home have high response latency with flow interval f_1 between 4-13s, which we analyze as being caused by proxy servers. While Xiaomi Home has the lowest response latency, with flow interval f_1 within 2s. Consequently, to achieve high detection performance on different platforms, the dynamic configurations of detector is necessary.

D. Performance of CRT Mining

We present the results of cross-rule interactions (CRT1-2) and interference (CRT3-6) in two parts and validate the effectiveness of CP-IoT by comparing it with IoTGaze and iRuler.

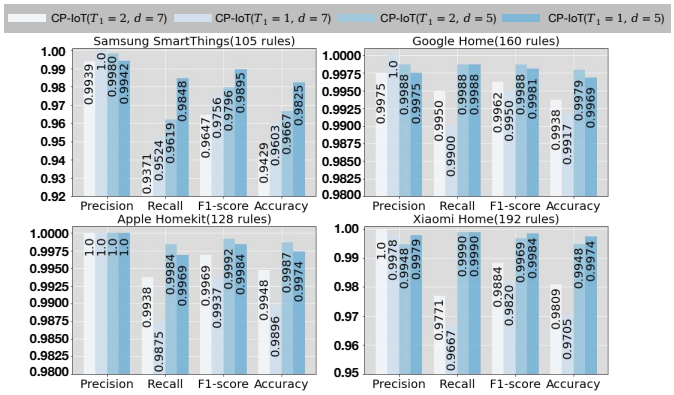


Fig. 14: SRA detection performance on four platforms.

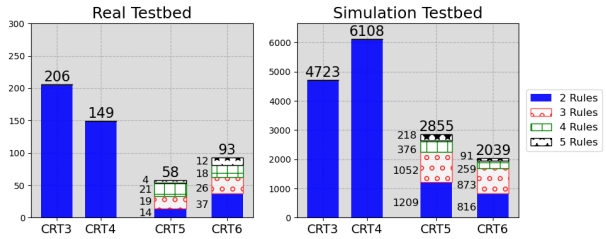


Fig. 15: The number of CRT3-6 discovered by CP-IoT.

Mining Results of CRT1-2. The discovered CRT1 can be verified directly since the result of the first rule execution directly triggers the second rule ($C_{R1} = E_{R2}$). And the result of CRT2 depends on the correctness of the physical correlation analysis ($C_{R1} \xrightarrow{?} E_{R2}$). We reproduce all physical correlations on the real testbed and remove the results that cannot be achieved. A typical case is illustrated in **Appendix D.C**, where we set the threshold $T_2 = 0.75$ discussed in **Appendix D.B**. We find that four channels *motion*, *sound*, *acceleration*, *airQuality* have most related correlations since many device actions can change their states. For example, $E^{fan/AC}_{switch.on}$ or $E^{window}_{window.open}$ will improve the air quality by promoting air circulation. We totally find 1171 CRT1 and 487 CRT2 on the real testbed, including 68.15% single-platform interactions and 31.85% cross-platform interactions. In the simulation testbed, we totally find 1072 CRT1 and 1461 CRT2.

Some examples of the results are presented in **Table.VI**. We find some high-risk interactions on the real testbed, such as turning on the fan after detecting smoke in **Case 4**, which accidentally trigger the thermostat to be in heating mode and continuous working could lead to a fire. **Case 6** and **Case 8** accidentally turns on the camera, which could cause privacy data leakage. On the simulation testbed, the rich device types and automation rules increase the exploration space of our mining methods. **Case 11** finds that the vacuum cleaner working could accidentally trigger the opening of the dishwasher, which increases the risk of flooding. Moreover, Rule1 of **Case 11** can also trigger the Rule1 of **Case 16** and indirectly open the window, which increase the risk of burglary.

Mining Results of CRT3-6. Since there are also interference and duplication between the rules of different platforms, we combine the rules deployed on the four platforms and analyze the cross-rule interference that exists among these rules. The identification results of CP-IoT are shown in **Fig.15**. For Action Conflict (CRT3) and Action Reverting (CRT4), we

TABLE VI: Some cases of cross-rule interactions mining results, where "R" denotes real testbed, "S" denotes simulation testbed, "P" denotes physical interaction and "C" denotes cyberspace interaction.

No.	Rule1	Rule2	Type	Testbed	Risk	No.	Rule1	Rule2	Type	Testbed	Risk
1	$E^{sensor}_{presence.present} \rightarrow C^{fan}_{switch.on}$	$E^{sensor}_{motion.active} \rightarrow C^{light}_{switch.on}$	P	R	Low	9	$E^{sensor}_{motion.active} \rightarrow C^{light}_{switch.on}$	$E^{sensor}_{illuminance.high} \rightarrow C^{curtain}_{windowShade.close}$	P	S	Low
2	$E^{sensor}_{vibration.active} \rightarrow C^{humidifier}_{switch.on}$	$E^{sensor}_{water.wet} \rightarrow C^{light}_{color.blue}$	P	R	Low	10	$E^{sensor}_{CO.detected} \rightarrow C^{siren}_{alram.siren}$	$E^{sensor}_{sound.high} \rightarrow C^{TV}_{volume.down}$	P	S	Low
3	$E^{sensor}_{motion.active} \rightarrow C^{humidifier}_{switch.on}$	$E^{sensor}_{humidity.high} \rightarrow C^{fan}_{switch.on}$	P	R	Low	11	$E^{sensor}_{dustLevel.high} \rightarrow C^{robot}_{switch.on}$	$E^{sensor}_{motion.active} \rightarrow C^{dishwasher}_{switch.on}$	P	S	High
4	$E^{sensor}_{smoke.detected} \rightarrow C^{fan}_{switch.on}$	$E^{sensor}_{temperature.low} \rightarrow C^{thermostat}_{mode.heat}$	P	R	High	12	$E^{sensor}_{humidity.low} \rightarrow C^{humidifer}_{switch.on}$	$E^{powerMeter}_{energy.high} \rightarrow C^{camera}_{switch.off}$	P	S	High
5	$E^{button}_{button.pressed} \rightarrow C^{thermostat}_{mode.heat}$	$E^{sensor}_{temperature.high} \rightarrow C^{fan}_{switch.on}$	P	R	Low	13	$E^{sensor}_{contact.closed} \rightarrow C^{TV}_{switch.off}$	$E^{sensor}_{sound.low} \rightarrow C^{camera}_{switch.on}$	P	S	High
6	$E^{sensor}_{temperature.low} \rightarrow C^{fan}_{switch.off}$	$E^{sensor}_{motion.inactive} \rightarrow C^{camera}_{switch.on}$	P	R	High	14	$E^{sensor}_{dustLevel.high} \rightarrow C^{robot}_{switch.on}$	$E^{sensor}_{presence.present} \rightarrow C^{lock}_{lock.unlocked}$	P	S	High
7	$E^{sensor}_{contact.open} \rightarrow C^{light}_{switch.on}$	$E^{light}_{switch.on} \rightarrow C^{light}_{color.blue}$	C	R	Low	15	$E^{sensor}_{illuminance.low} \rightarrow C^{light}_{switch.on}$	$E^{light}_{switch.on} \rightarrow C^{curtain}_{windowShade.open}$	C	S	Low
8	$E^{Time}_{time.night} \rightarrow C^{Mode}_{mode.night}$	$E^{Mode}_{mode.night} \rightarrow C^{camera}_{switch.on}$	C	R	High	16	$E^{sensor}_{motion.active} \rightarrow C^{Mode}_{mode.home}$	$E^{Mode}_{mode.home} \rightarrow C^{window}_{window.open}$	C	S	High

explore the entire rule-pair space $O(C(N,2))$, and the number of mining on both testbeds is substantial. We find that the most of rules associated with devices have CRT2 and that most of the CRT1 occur under two rules in different scenes. Action Reverting (CRT5) and Action Loop (CRT6) are represented as directed loops on the graph. We only explore part of the rule combination space that consider the rule chains contain no more than **5** rules, since the identification time overhead grows geometrically with limited results and iRuler finds most of CRT5,6 contains 2-3 rules.

Mining Results Comparison. We use two state-of-the-art models to compare the performance of CRT mining with CP-IoT, which is tested on the simulation testbed as these two models do not support cross-platform mining. IoTGaze uses 8 cyberspace channels and 9 physical channels. Since iRuler has implicit introduction of CRT1-2 mining method, we use CP-IoT's mining results to construct intermediate representations (IR) and discover CRT3-6. The result is shown in **Table VII**. For CRT1-2, we consider the richer physical channels and more complete shared common capability than IoTGaze. Our findings basically cover the mining results of both two baselines and discover slightly more CRT3-6 than iRuler, as both of them perform a complete searching of the rule combination space but CP-IoT considers more feasible rule chains triggered by multiple physical interactions.

TABLE VII: Comparison of the mined number of CRT in the simulation testbed.

Number / Method / Kind	IoTGaze	iRuler	CP-IoT
Physical Interactions(CRT1)	827	N/A	1461
Cyberspace Interactions(CRT2)	344	N/A	1072
Action Conflict(CRT3)	N/A	4619	4723
Action Duplicate(CRT4)	N/A	6025	6108
Action Reverting(CRT5)	N/A	2704	2855
Action Loop(CRT6)	N/A	1856	2039

VIII. Related Work

With the rapid development of IoT, the security issues about home automation have been widely researched [14]–[20], [25], [55] in recent years. These method can be coarsely classified into two categories. The first one is to check the consistency of automation rules and monitor whether an individual rule is executed correctly. Homonit [14] and IoTGaze [15] are two similar systems that model the extracted rules and detect the unexpected behaviors by context checking. HAWatcher [16] extracts correlation between devices based on various semantic information such as logs, device configurations, etc. Its shadow execution engine can detect deviations between the runtime device behavior and correlation.

The other type is to check the reliability of communication channels whether the deployed automation interacts with each other unexpectedly through certain channels or generates interference. Homeguard [20] builds a symbolic execution module to extract automation semantics from applications and identify three application interference threats. IoTGuard [18] designs a dynamic system that collects the runtime information about the application and stores it in a dynamic model. It identifies app information that violates security policies. iRuler [19] formalizes the rules and applies SMT and model checking technique to discover inter-rule vulnerabilities.

IX. Discussion

Limitations. After an experienced attacker masters the communication pattern between the device and platform, the replayed event packet cannot be detected by CP-IoT since it does not violate the event fingerprint (such as MitM attack). The second limitation is that the space complexity of CRT Mining is $O(n^2)$, which costs a lot of time in large-scale scenarios. Finally, detecting event losses and some command failure cases requires manual effort to determine the device state by obtaining logs, which requires developer permissions.

Ethic Consideration. Considering that automation rule extraction requires app pages and may violate users' privacy, prior consent needs to be sought from users. Moreover, the anomaly detection requires intercepting communication traffic via side channel, which may contain user's privacy. Our approach only saves the packet sequence involving the rule execution part. Finally, the cross-rule risks we find may cause significant security risks. To prevent attackers exploitation, we use the user's device id to represent each IoT device, since it is very difficult to obtain the mapping relationship between all devices and IDs within different scenarios.

X. Conclusion

Influenced by market size and user preferences, different IoT platforms are used in different regions around the world. We propose a cross-platform monitoring system CP-IoT for smart home, which is capable of finding single-platform and cross-platform threats. We propose an app-page-based rule extraction method and a multi-granularity rule behaviour identification technique, which can address architecture and traffic differences between platforms. We construct a centralised graph to portray device state changes and rule execution across platforms. Based on this, we design two algorithms to discover various rule execution anomalies and cross-rule threats, and propose the mitigation for these threats. We validate CP-IoT on four mainstream IoT platforms and the results demonstrate the effectiveness of CP-IoT in detecting various threats and excellent cross-platform compatibility.

ACKNOWLEDGEMENT

We thank our shepherd and all the anonymous reviewers for their valuable comments to improve this paper. This work is supported by the National Key R&D Program of China under Grant 2022YFB3102902, and the National Natural Science Foundation of China (No. 62172251).

REFERENCES

[1] Amazon alexa, 2023. https://alexa.amazon.com/.

[2] Samsung smartthings, 2023. https://www.samsung.com/us/smartthings.

[3] Apple homekit, 2023. https://www.apple.com.cn/apple-home/.

[4] Google home, 2023. https://home.google.com/welcome/.

[5] Xiaomi home, 2023. http://home.mi.com/.

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

[7] Blase Ur, Elyse McManus, Melwyn Pak Yong Ho, and Michael L Littman. Practical trigger-action programming in the smart home. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 803–812, 2014.

[8] Samsung smartthings community, 2023. https://community.smartthings.com/c/smartapps/6.

[9] Google nest community, 2023. https://www.googlenestcommunity.com/.

[10] Home assistant community, 2023. https://community.home-assistant.io/latest.

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

[12] Atheer Abu Zaid, Manar H. Alalfi, and Ali Miri. Automated identification of over-privileged smartthings apps. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 247–251, 2019.

[13] Earlence Fernandes, Jaeyeon Jung, and Atul Prakash. Security analysis of emerging smart home applications. In *2016 IEEE symposium on security and privacy (SP)*, pages 636–654. IEEE, 2016.

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

[15] Tianbo Gu, Zheng Fang, Allaukik Abhishek, Hao Fu, Pengfei Hu, and Prasant Mohapatra. Iotgaze: Iot security enforcement via wireless context analysis. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, pages 884–893. IEEE, 2020.

[16] Chenglong Fu, Qiang Zeng, and Xiaojiang Du. Hawatcher: Semantics-aware anomaly detection for appified smart homes. In *30th USENIX Security Symposium (USENIX Security 21)*, 2021.

[17] Z Berkay Celik, Patrick McDaniel, and Gang Tan. Soteria: Automated iot safety and security analysis. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pages 147–158, 2018.

[18] Z Berkay Celik, Gang Tan, and Patrick D McDaniel. Iotguard: Dynamic enforcement of security and safety policy in commodity iot. In *NDSS*, 2019.

[19] Qi Wang, Pubali Datta, Wei Yang, Si Liu, Adam Bates, and Carl A Gunter. Charting the attack surface of trigger-action iot platforms. In *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*, pages 1439–1453, 2019.

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

[21] Chandrakana Nandi and Michael D Ernst. Automatic trigger generation for rule-based smart homes. In *Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security*, pages 97–102, 2016.

[22] Iulia Bastys, Musard Balliu, and Andrei Sabelfeld. If this then what? controlling flows in iot apps. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, pages 1102–1119, 2018.

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

[24] Wenbo Ding, Hongxin Hu, and Long Cheng. Iotsafe: Enforcing safety and security policy with real iot physical interaction discovery. In *the 28th Network and Distributed System Security Symposium (NDSS 2021)*, 2021.

[25] Yuan Tian, Nan Zhang, Yue-Hsun Lin, XiaoFeng Wang, Blase Ur, Xianzheng Guo, and Patrick Tague. Smartauth: User-centered authorization for the internet of things. In *USENIX Security Symposium*, volume 5, pages 8–2, 2017.

[26] Xuan Feng, Qiang Li, Haining Wang, and Limin Sun. Acquisitional rule-based engine for discovering internet-of-things devices. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 327–341, 2018.

[27] Z Berkay Celik, Leonardo Babun, Amit Kumar Sikder, Hidayet Aksu, Gang Tan, Patrick McDaniel, and A Selcuk Uluagac. Sensitive information tracking in commodity iot. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1687–1704, 2018.

[28] Huan Feng, Kassem Fawaz, and Kang G Shin. Continuous authentication for voice assistants. In *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking*, pages 343–355, 2017.

[29] Kyriakos Georgiou, Samuel Xavier-de Souza, and Kerstin Eder. The iot energy challenge: A software perspective. *IEEE Embedded Systems Letters*, 10(3):53–56, 2017.

[30] Farid Molazem Tabrizi and Karthik Pattabiraman. Design-level and code-level security analysis of iot devices. *ACM Transactions on Embedded Computing Systems (TECS)*, 18(3):1–25, 2019.

[31] Munish Bhatia, Simranpreet Kaur, and Sandeep K Sood. Iot-inspired smart home based urine infection prediction. *Journal of Ambient Intelligence and Humanized Computing*, pages 1–15, 2020.

[32] Maissa Dammak, Omar Rafik Merad Boudia, Mohamed Ayoub Messous, Sidi Mohammed Senouci, and Christophe Gransart. Token-based lightweight authentication to secure iot networks. In *2019 16th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, pages 1–4. IEEE, 2019.

[33] Yunhan Jack Jia, Qi Alfred Chen, Shiqi Wang, Amir Rahmati, Earlence Fernandes, Zhuoqing Morley Mao, Atul Prakash, and SJ Unviersity. Contexiot: Towards providing contextual integrity to appified iot platforms. In *ndss*, volume 2, pages 2–2. San Diego, 2017.

[34] Vincent F Taylor, Riccardo Spolaor, Mauro Conti, and Ivan Martinovic. Appscanner: Automatic fingerprinting of smartphone apps from encrypted network traffic. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 439–454. IEEE, 2016.

[35] Parth Bhatt and Anderson Morais. Hads: Hybrid anomaly detection system for iot environments. In *2018 international conference on internet of things, embedded systems and communications (IINTEC)*, pages 191–196. IEEE, 2018.

[36] Stanfordcorenlp tool, 2022. https://github.com/stanfordnlp/CoreNLP.

[37] Samsung smartthings developer documentation, 2022. https://developer.smartthings.com/docs/devices/capabilities/capabilities-reference.

[38] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[39] John A Hartigan and Manchek A Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the royal statistical society. series c (applied statistics)*, 28(1):100–108, 1979.

[40] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

[41] Toyota smart home dataset. https://project.inria.fr/toyotasmarthome/.

[42] Iot analytics benchmark dataset, 2023. https://github.com/vmware-archive/iot-analytics-benchmark.

[43] Smart* data set, 2023. https://traces.cs.umass.edu/index.php/smart/smart.

[44] Ifttt official website, 2023. https://ifttt.com/explore/applets.

[45] Smartthings smartapp public repository, 2022. https://github.com/SmartThingsCommunity/SmartThingsPublic.

[46] Neo4j graph database. https://neo4j.com.

[47] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[48] Google news dataset. https://github.com/mmihaltz/word2vec-GoogleNews-vectors.

[49] Wifi attack tools: aircrack-ng. https://www.kali.org/tools/aircrack-ng/.

[50] Zigbee attack tools: Killerbee. https://github.com/riverloopsec/killerbee.

[51] Zigbee attack tools: Z-attack. https://github.com/initbrain/Z-Attack.

[52] Bluetooth attack tools: btlejack. https://github.com/virtualabs/btlejack.

[53] Paul E Black. Levenshtein distance, dictionary of algorithms and data structures [online], us national institute of standards and technology, 2008.

[54] Bluetooth packet capture tool. https://linux.die.net/man/8/hcidump.

[55] 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 *2018 48th annual IEEE/IFIP international conference on dependable systems and networks (DSN)*, pages 610–621. IEEE, 2018.

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

[57] Create fake events for iot devices., 2022. https://community.smartthings.com/t/create-events-for-iot-devices/242952.

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

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

[60] Smartcam motion event interception, 2016. https://community.smartthings.com/t/use-smartcams-motion-sensor-with-smartthings/54364.

[61] Adrienne Porter Felt, Erika Chin, Steve Hanna, Dawn Song, and David Wagner. Android permissions demystified. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 627–638, 2011.

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

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

[64] Kulani Mahadewa, Yanjun Zhang, Guangdong Bai, Lei Bu, Zhiqiang Zuo, Dileepa Fernando, Zhenkai Liang, and Jin Song Dong. Identifying privacy weaknesses from multi-party trigger-action integration platforms. In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 2–15, 2021.

[65] Musard Balliu, Iulia Bastys, and Andrei Sabelfeld. Securing iot apps. *IEEE Security & Privacy*, 17(5):22–29, 2019.

[66] Eyal Ronen and Adi Shamir. Extended functionality attacks on iot devices: The case of smart lights. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 3–12. IEEE, 2016.

[67] Musard Balliu, Massimo Merro, Michele Pasqua, and Mikhail Shcherbakov. Friendly fire: cross-app interactions in iot platforms. *ACM Transactions on Privacy and Security (TOPS)*, 24(3):1–40, 2021.

APPENDIX A
THREAT PLANE COVERED BY CP-IOT

Single-Rule Anomalies(SRA) occur when the runtime behavior of an automation rule is inconsistent with its specification. Specifically, we classify them into the following types and list some real-world cases.

- **SRA1: Faulty Event/ Fake Event**. Faulty Event is made by the component defect of IoT devices or channel interference that report a event incorrectly, such as the faulty door knocking event send by the SmartThings Multisensors [56] and the ghost motion event reported by the Fibrao Motion Sensor [11]. Besides, an attacker can inject a specific event using the side channel to trigger an automation rule, which results in Fake Event anomalies [57], [58].
- **SRA2: Event Losses/ Event Interception**. This type of exception occurs when events are not received by the platform properly. The former can be caused by network delays or IoT Cloud side failures, such as the update delay of the presence event send by the SmartThings Arrival Sensor [59] while the latter is caused by an attacker intercepting the event packets [60].
- **SRA3: Over-Privilege**. This anomaly have been widely discussed in the previous studies [13], [61]. Specifically, an over-privilege anomaly occurs in the following two cases: (1) the automation rule controls unrelated devices and (2) the rule controls devices to certain unrelated states.
- **SRA4: Command Failure/ Command Interception**. Some commands are sent to devices when the events are received by IoT Cloud correctly. Similar to Event Interception, the command packets can also be intercepted by attackers [62]. Besides, the large network delay, system crash or the device component breakdown will make commands fail to take effect. SmartThings Power Outlet [62] and TP-Link HS110 Smart Wi-Fi Plug [63] have been reported to have such anomalies.

In contrast, the CRT are more complicated and occur when a user deploys multiple automation rules. One of the cases is that an rule ends its execution and accidentally triggers another rule, where we call **cross-rule interactions**. Specifically, they can be summarized into the following two types:

- **CRT1: Cyberspace Interactions** [17], [27]. For two automations rules sharing the same device or the same event/command to interact, we call cyberspace interaction. An example is depicted in **Fig.2**, a rule turns on the light($C^{light}_{switch.on}$) when the time is at sunset($E^{Time}_{time.sunset}$), and another rule unlocks the door($C^{lock}_{lock.unlock}$) when the light is on($E^{light}_{switch.on}$). These two rules interact through "light.on" event.
- **CRT2: Physical Interactions** [18], [23]. When one automation is executed completely, the result may change the physical environment and implicitly trigger the execution of another automation. For example, a rule turns on the air conditioning($C^{AC}_{switch.on}$) when detects the user at home($E^{sensor}_{presence.present}$). Another rule turns on the heater ($E^{heater}_{switch.on}$) when the temperature is below a threshold ($C^{sensor}_{temperature.low}$). The command ($C^{AC}_{switch.on}$) of the first rule changes the temperature, which accidentally triggers the second rule.

Another type of CRT occurs when two rules interfere with each other, which we call **cross-rule interference**. We summarize the following four subclasses of this type.

- **CRT3: Action Conflict** [19], [20], [64], [65]. This refer to the command part of two rules conflict with each other, where they change the same device to different states. As shown in **Fig.2**, one rule turns off the fan($C^{fan}_{switch.off}$) when the time is night($E^{Time}_{time.night}$), and another rule turns on the fan($C^{fan}_{switch.on}$) when the sensor detects user activity($E^{sensor}_{motion.active}$). Both automations

seem feasible when the user firstly deploys them. However, they meets a conflict accidentally when both trigger conditions($E^{Time}_{time.night}, E^{sensor}_{motion.active}$) are satisfied at the same time.

- **CRT4: Action Duplicate** [17]–[19]. Such conflicts may occur when two rules perform the same action on the same device. A typical example is shown in **Fig.2** where two automation rules turn on the light repeatedly.
- **CRT5: Action Reverting** [19], [33], [66]. Similar to Action Conflict, there is a conflict between two rules. The difference is that there is a trigger chain of automations between two conflicting rules. We show an example in **Fig.2**, one rule opens the curtain($C^{curtain}_{curtain.open}$) and turns on the light($C^{light}_{switch.on}$) when user arrives home($E^{sensor}_{presence.present}$), and another rule closes the curtain ($C^{curtain}_{curtain.closed}$) when the sensor detects high illuminance($E^{sensor}_{illuminance.high}$). The actions of these two rules are conflict and the first rule triggers the second rule through the physical interaction.
- **CRT6: Action Loop** [19], [67]. The action of the first rule directly triggers the second rule, and the action of the second rule triggers the first rule, leading to a dead loop. In the real smart home, this threat generally occurs in cross-scenario situations or related to the system mode.

APPENDIX B
SUPPLEMENTARY FOR THE SYSTEM DESIGN

Algorithm 1: SRA detection algorithm.

procedure: $S_l \leftarrow Similarity(S_c, S_e, S_{log}, \mathcal{G}_{cs}, \mathcal{G})$
$S_l \leftarrow [], \mathcal{P}_{id} \leftarrow \phi, l_1 \leftarrow len(S_c), l_2 \leftarrow len(S_e)$;
if $S_c = \phi$ && $S_e = \phi$ && $Change\_State(S_{log})$ **then**
  $return$ "$Anomaly : SRA2$" ;
for $i = 1$ *to* l_1, $j = 1$ *to* l_2 **do**
  $\mathcal{P}_{id} \leftarrow \mathcal{P}_{id} \cup Find\_Rule(S_c[i], \mathcal{G})$;
  $\mathcal{P}_{id} \leftarrow \mathcal{P}_{id} \cup Find\_Rule(S_e[j], \mathcal{G})$;
while $\mathcal{P}_{id} \neq \phi$ **do**
  $r_i \leftarrow Pop(\mathcal{P}_{id})$;
  $G_{sp} \leftarrow Find\_Path(r_i)$;
  $s_e, s_c, s_{total} \leftarrow GraphSAGE(\mathcal{G}_{cs}, \mathcal{G}_{sp})$;
  $S_l.append((s_{total}, s_e, s_c))$;
if $\forall s \; in \; S_l, \; s[0] \neq 1$ **then**
  Report "$Anomaly : SRA1/3/4$" ;
else
  Report "$Normal$" ;

A.Single-Rule Anomalies Detection Algorithm

Algorithm.1 shows the workflow of detecting the anomalies in the rule execution. **Lines 4-9** correspond to the first stage of SRA detection method, **specification matching**. In this stage, we find the specification graph in the centralised model constructed in Section V-B that is most similar to the running rule. **Lines 2-3** and **lines 10-15** correspond to the second stage of SRA detection method, **consistency checking**. In this phase, we match the behavior graph of the rule execution with the specification graph extracted in the first stage and check whether there are inconsistencies to find anomalies.

Algorithm 2: CRT1-2 inference algorithm.

procedure: $S_{cyb}, S_{phy} \leftarrow$ Inference($\mathcal{G}$)
1 $S_{cyb} \leftarrow \phi, \; S_{phy} \leftarrow \phi, \; S_{id} \leftarrow Find\_Rule(\mathcal{G})$;
2 **while** $S_{id} \neq \phi$ **do**
3 $r_{cur} \leftarrow Pop(S_{id})$;
4 $S_{com} \leftarrow Combination(r_{cur}, S_{id})$;
5 **while** $S_{com} \neq \phi$ **do**
6 $r_1, r_2 \leftarrow Pop(S_{com})$;
7 **if** $r_1.commands \supseteq r_2.events$ **then**
8 $S_{cyb} \leftarrow S_{cyb} \cup (r1, r2)$;
9 **if** $r_2.commands \supseteq r_1.events$ **then**
10 $S_{cyb} \leftarrow S_{cyb} \cup (r2, r1)$;
11 $E_{r1} \leftarrow Transform(r_1.commands)$;
12 $E_{r2} \leftarrow Transform(r_2.commands)$;
13 **if** $E_{r1} \supseteq r_2.events$ **then**
14 $S_{phy} \leftarrow S_{phy} \cup (r1, r2)$
15 **if** $E_{r2} \supseteq r_1.events$ **then**
16 $S_{phy} \leftarrow S_{phy} \cup (r2, r1)$

B. Cross-Rule Interactions Inference Algorithm

Algorithm.2 shows the process of searching cross-rule interactions on the graph. We firstly find all the rules ($Find\_Rule$) on the graph and save them in the set S_{id}. Afterwards we combine each element of the set with other rules ($Combination$) and determine whether exists a cross-rule interactions between two rules. **Lines 7-10** of the algorithm search all cyberspace interactions on the graph $\mathcal{G}$. If the command set of a rule ($r_1.commands$) contains the event set of another rule ($r2.events$), the constraint is satisfied and exists a cyberspace interaction between them. **Lines 11-16** of the algorithm search all physical interactions on the graph $\mathcal{G}$. Using the $transform$ method, we get the event set (E_{r1}, E_{r2}) associated with the commands of an rule through the physical channel, and determine if event sets contain all the events needed to trigger another rule.

C. Dangerous Actions Caused by Cross-Rule Interactions

We describe five security issues and the associated commands in **Table.VIII**. For each of the two rules involving CRT1/2, the interactions will be considered as high risky if the triggered rule performs an action in this list. Apart from these five dangers, excessive power consumption is a concern for many users. The usual culprits are the constant work of energy-intensive devices such as lights, TVs, and air conditioners. We exclude them in the high-risk list since they cause limited damage to the home environment.

TABLE VIII: A list of some high-risk actions.

Action	Security Issue	Action	Security Issue
$C^{heater}_{switch.on}$	Fire	$C^{thermostat}_{mode.heat}$	Fire
$C^{thermostat}_{mode.cool}$	Chilly	$C^{AC}_{switch.on}$	Chilly
$C^{faucet/dishwasher}_{switch.on}$	Water Flooding	$C^{valve}_{valve.open}$	Water Flooding
$C^{window}_{window.open}$	Burglary	$C^{lock}_{lock.unlocked}$	Burglary
$C^{camera}_{switch.on}$	Privacy Leakage	$C^{soundbox}_{switch.on}$	Privacy Leakage

Fig. 16: 32 devices deployed in our lab for experiment.

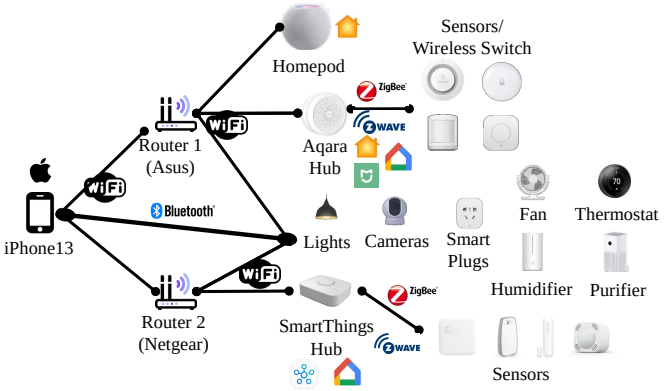


Fig. 17: The device deployment topology of the real testbed.

APPENDIX C
SUPPLEMENTARY FOR THE EXPERIMENTAL SETUP

A. The Real Testbed

32 IoT devices deployed in our lab are shown in **Fig.16** and the layout of the device is shown in **Fig.17**. To achieve a low-latency network environment, we use two routers with 1000Mbps bandwidth, respectively are NETGEAR router (Netgear 35) and ASUS router (NMG24_2). For the SmartThings and Google Home, we use SmartThings Hub as the gateway. For Apple Homekit and Xiaomi Home, we use Aqara Hub MS1 as the gateway to connect some small sensors. Moreover, multiple rules are deployed on devices that support different platforms to achieve a richer experimental test environment. The details of each device are listed in our open source project.

B. The Simulation Testbed

The simulation testbed layout is shown in **Fig.18**. We simulate a smart home environment with seven scenarios, including entrance, living room, bedroom, kitchen, dining room, balcony and bathroom. In each scenarios, we simulate the deployment of various common-used types of smart home devices, such as sprinkler, dishwasher and robot cleaner. The details of each device are listed in our open source project.

C. Anomaly Cases Generation

Here we describe methods for generating SRA1-SRA4.

Fake Events: An attacker will capture packets of certain event when a rule is executed and injects malicious fields into the packets to control a device. We collect the event traffic for each device, add 2-7 bytes of zeros to the payload of the last packet and replay these packets.

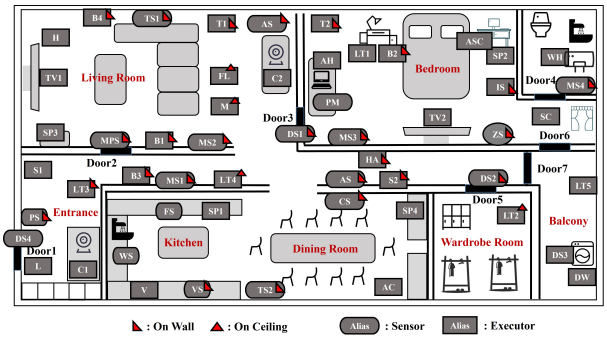


Fig. 18: The simulation testbed.

Event Losses: An attacker may disable the event portion of a rule's execution by intercepting packets, resulting in the rule not being executed. So we collect the execution traffic for each rule, remove all packets belong to the event part and replay the rest of the packets.

Command Failure-Cyberspace: An attacker may disable the command portion of a rule's execution by intercepting packets, resulting in the rule not being executed. We collect the execution traffic for each rule, remove all packets belong to the command part and replay the rest of the packets.

Command Failure-Physical: This type is caused by device failure. We cut the power of the device and trigger the rule to simulate the device defect.

Over-Privilege: We collect the normal execution traffic of the rules and identify the events and the command part C_1. Then we construct the dataset of this anomaly type in two ways: (1) Add the packet sequence of a previously collected command C_2 to the command section, C_1 and C_2 control different devices. (2) Remove all packets from the command section and add the packet sequence of a previously collected command C_2, where C_1 and C_2 control the same device, but with opposite actions such as "switch.on" and "switch.off".

APPENDIX D
SUPPLEMENTARY FOR THE EVALUATION

A. Time Overhead for Model Extension

We evaluate the time cost for adding each device deployed in the real testbed. We remove all information about this device from the model, including associated rules, graph nodes and CRT mining results. Then we remove the device and all related automation rules on the app. After that we re-add the devices in one supported app and add five automation rules for each

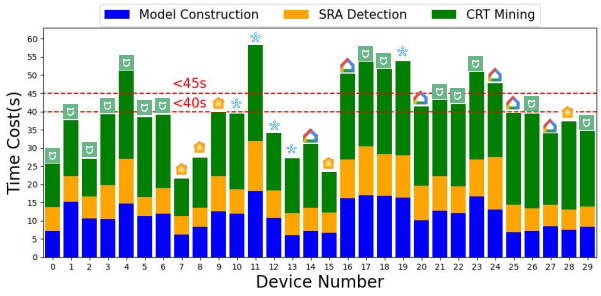
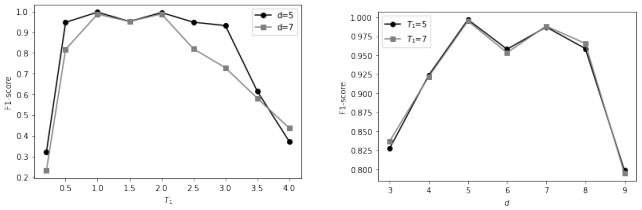


Fig. 19: The time overhead for adding a new device.

device, since most devices are associated with only a small number of rules. Finally, we calculate the running time for each part and the total time.

As shown in **Fig.19**, a simple device such as most sensors and smart outlets, the total time cost is less than **40 s**. Some sensors have more complex functionality such as the Aeotec Multipurpose Sensor have six functions, but the overhead of extension are all within **45 s**. In particular, the model construction comprises two parts: rule extraction and graph building. For some more complex devices such as camera and color lamps, the total time overhead is less than **1 min**. The increased time comes from CRT Mining, since they are associated with more CRT. The time to add a platform depends on the devices added, and rule extraction is not slowed down by the devices in the new platform.

B. Sensitivity of Experimental Results to Hyper-parameters



(a) The effect of changes in the parameters $T_1(fixed\ d)$ on F1-score. (b) The effect of changes in the parameters $d(fixed\ T_1)$ on F1-score.

Fig. 20: The sensitivity of the SRA detection results to variations in the parameters T_1, d.

Fig.20(a) show the effect of variation in the parameters $T_1(fixed\ \ d)$ on the F1-score judging the SRA detection performance. Similarly, **Fig.20(b)** show the effect of variation of parameter $d(fixed\ T_1)$ on the F1-score.

The parameter T_2 is related to the clause embedding transformer, which affects the value of the calculated cosine similarity. When its value is too large, all possible event correlations through the physical channel are not considered and make the number of CRT mining results drastically reduced. However, when the parameter T_2 value is too small, many physical correlations are incorrectly determined since two events with low correlation scores are also judged as physically-correlated, resulting in many false CRT results.

C. Security Implication of Discovered Threats

As described in the threat model in **section III-C**, anomalies in rule execution may be caused by attackers. Here we illustrate the security impact caused by SRA through two examples of injected attacks in real testbed. The first one is to replay the **fake events** attack. The rule listed in **Table.IX** turns on the fan when executed correctly. We inject 2 all-0 bytes in the first packet of event $E_{motion.active}$ and replay it into the environment. It will cause the rule to be executed and generate traffic to control the smart fan. The other is to replay the **over-privilege** attack. We capture the traffic of other device events/commands and inject it into the original traffic using two ways. This is used to simulate an attacker illegally elevating the rule privileges to control other unrelated devices(Mi Desk Lamp). CP-IoT can detect both by matching fine-grained fingerprint information with the runtime traffic and find the inconsistencies between them.

TABLE IX: Two SRA attack vectors injected on a rule deployed on the real testbed.

Attack	Vector	Devices
Normal $E^{sensor}_{motion.active} \rightarrow C^{fan}_{switch.on}$	[(113, 0), (65, 1), (112, 0), (65, 1)] → [(54, 1), (63, 0), (40, 1), (40, 1)]	Aqara Motion Sensor Mi Smart Fan
Fake Event Inject **"0x0000"** at the end of the payload of the packet(113, 0)	[(115, 0), (65, 1), (112, 0), (65, 1)] → [(54, 1), (63, 0), (40, 1), (40, 1)]	Aqara Motion Sensor
Over-Privilege Inject the traffic generated by the command $C^{lamp}_{switch.on}$ to the traffic	[(115, 0), (65, 1), (112, 0), (65, 1)] → [(54, 1), (63, 0), (40, 1), (40, 1), (227, 1), (28, 0), (195, 0), (28, 1), (96, 1), (28, 0)]	Mi Smart Desk Lamp 1s
	[(115, 0), (65, 1), (112, 0), (65, 1)] → (227, 1), (28, 0), (195, 0), (28, 1), (96, 1), (28, 0), (54, 1), (63, 0), (40, 1), (40, 1)]	

TABLE X: A cross-platform CRT2 discovered by the CP-IoT.

Rule	Part	Description	Symbol
Rule1	Event/ Trigger Condition	Time at night 22pm-7am	$E^{Time}_{time.night}$
		No human presence	$E^{sensor}_{presence.notpresent}$
		Each 5min interval	$E^{Time}_{time.interval}$
	Command/ Action	Open the camera	$C^{camera}_{switch.on}$
		Set the mode at surveillance	$C^{Time}_{mode.survilleance}$
Rule2	Event/ Trigger Condition	Detects the motion	$E^{sensor}_{motion.active}$
	Command/ Action	Open the thermostat	$C^{thermostat}_{switch.on}$
		Set the mode at heat	$C^{thermostat}_{thermostatMode.heat}$

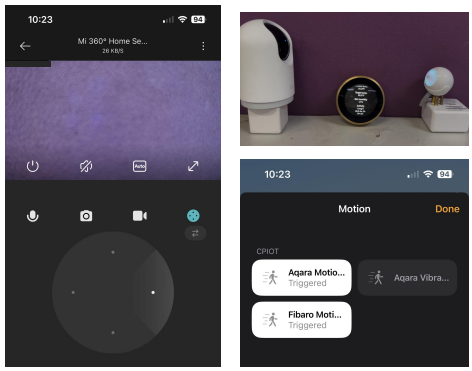


Fig. 21: The reproduction result of a physical interaction.

Cross-platform threats are special types of CRT that involve rule interactions across multiple platforms. Here we conduct a case study of CRT found by CP-IoT on the real testbed and describe the results reproduced in our lab.

The CRT shown in **Table.X** involve two platforms, Xiaomi Home and Homekit. The first rule turns on the camera at night to monitor the home environment and will perform a 360-degree scan every 5 minutes, which will indirectly triggers some motion sensors to detect the motion. We reproduce this CRT2 case in our lab and show this physical interaction process in **Fig.21**. It is worth pointing out that of the four cameras we tested only this one triggers a sensor false detection. This is due to the fact that most of the cameras have a relatively limited magnitude and speed of rotation in the surveillance mode. This action implicitly triggers another rule, which set the Google thermostat at the heating mode (this rule usually set at the winter), where its constant heating can lead to a fire.