On the Safety of IoT Device Physical Interaction Control

Wenbo Ding
Clemson University
wding@clemson.edu

Hongxin Hu
Clemson University
hongxih@clemson.edu

ABSTRACT

Emerging Internet of Things (IoT) platforms provide increased functionality to enable human interaction with the physical world in an autonomous manner. The physical interaction features of IoT platforms allow IoT devices to make an impact on the physical environment. However, such features also bring new safety challenges, where attackers can leverage stealthy physical interactions to launch attacks against IoT systems. In this paper, we propose a framework called IoTMon that discovers any possible physical interactions and generates all potential interaction chains across applications in the IoT environment. IoTMon also includes an assessment of the safety risk of each discovered inter-app interaction chain based on its physical influence. To demonstrate the feasibility of our approach, we provide a proof-of-concept implementation of IoTMon and present a comprehensive system evaluation on the Samsung SmartThings platform. We study 185 official SmartThings applications and find they can form 162 hidden inter-app interaction chains through physical surroundings. In particular, our experiment reveals that 37 interaction chains are highly risky and could be potentially exploited to impact the safety of the IoT environment.

KEYWORDS

Safety; Internet of Things; Physical Interaction Control

ACM Reference Format:
Wenbo Ding and Hongxin Hu. 2018. On the Safety of IoT Device Physical Interaction Control. In *2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18), October 15–19, 2018, Toronto, ON, Canada.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3243734.3243865

1 INTRODUCTION

The rapid development of Internet of Things (IoT) technologies brings true smart homes closer to reality. Nowadays, home automation has made a significant impact on the world economy, which is expected to reach $79 billion in 2022 according to the MarketsandMarkets [3]. Many commercial IoT platforms, such as Samsung's SmartThings [37], Apple's HomeKit [12], Wink [43], and Google Home [23], are readily available on the market. Other open source IoT platforms, such as openHAB [32] and IoTivity [26], have also emerged. Typically, these platforms have a hub controller to manage remote IoT devices, such as bulbs, cameras, and locks, and use applications (a.k.a. appified IoT platforms) to manage devices in an unattended manner, for example, turning on lights when users return home, monitoring users' kids from afar, or locking home doors while users drive away [38].

With the increased deployment of IoT devices, their security and safety problems have recently attracted significant attention [25, 35]. For example, by exploiting over 600,000 vulnerable IoT devices (using common factory default usernames and passwords), a large-scale Distributed Denial of Service (DDoS) attack was launched by the Mirai malware, which caused a massive Internet outage [5, 42]. By exploiting a firmware flaw and using a malicious mobile application, an attacker could conduct a multi-step attack to compromise a local home network from the Internet [36]. It was also reported that, by exploiting vulnerabilities in communication protocols, a worm could exploit flaws in ZigBee [2] to spread among smart bulbs [34]. In addition, researchers have recently found design flaws in Samsung's SmartThings platform, which allow malicious third-party applications to compromise the SmartThings platform [18]. Other researchers have also explored the possibility of utilizing IoT devices' physical capabilities to conduct attacks and demonstrated that a compromised smart bulb could sniff sensitive intranet information and send it out by flashing the light stealthily [1].

Despite considerable recent research on improving IoT security, existing research efforts have mainly focused on addressing *traditional* security issues in the IoT environment, such as device firmware bugs [21, 36], communication protocol vulnerabilities [22, 29, 34], malicious applications [18, 36], and system design flaws [18, 20, 27, 44]. Distinctive from existing work, our study reveals a *new* type of security problem that could happen due to the specific features of IoT platforms. One such feature is the ability for IoT devices to interact with their surroundings through *physical* interaction capabilities. Although such physical interactions of IoT devices could bring significant convenience to end users, they could also be potentially exploited by attackers to jeopardize IoT environments. The physical interaction capabilities enable devices to interact with each other through shared physical environments, such as air, temperature, and humidity. Since IoT applications manage IoT devices on most existing platforms, an application that controls devices to change physical environments may trigger certain executions of other applications. As a result, if the application is not aware of all of its possible interactions with other applications, some unexpected interactions could be exploited and triggered by attackers. For example, suppose that an attacker has obtained the access to a heater in an IoT network, which has installed a temperature-related application [39] that can open windows when the home temperature is higher than a given threshold. After turning on the heater for a period, the attacker can trigger the window opening action and cause a potential problem of break-in.

In this paper, we propose a framework called IoTMon that can capture all potential *physical* interactions across applications and

CCS '18, October 15–19, 2018, Toronto, ON, Canada

ACM ISBN 978-1-4503-5693-0/18/10…$15.00
https://doi.org/10.1145/3243734.3243865

enable safe interaction controls on IoT platforms. To address the problems caused by unexpected physical interactions, IoTMon first performs an intra-app interaction analysis using static program analysis to extract necessary application information, including triggers, devices, and actions, for building intra-app interactions. In addition to the static analysis of applications, IoTMon also uses Natural Language Processing (NLP) techniques to analyze application descriptions to identify physical channels on the IoT platform, and then connect intra-app interactions through physical and system channels to generate inter-app interaction chains. After identifying all interaction chains, IoTMon uses a risk analysis mechanism to evaluate the risk of identified inter-app interaction chains. Our evaluation based on 185 official SmartThings applications shows that 162 hidden interaction chains exist among these applications, and 37 of them are highly risky and could be potentially exploited. To the best of our knowledge, IoTMon provides the *first solution* to identify and analyze hidden interaction chains among IoT applications, enabling the safe control of IoT device interactions.

The rest of the paper is organized as follows. Section 2 presents threat model and problem scope. Section 3 gives a system overview of IoTMon. Section 4 introduces the details about IoTMon design and implementation. We describe the evaluation of IoTMon in Section 5. Related work is discussed in Section 6 and Section 7 concludes our work.

2 THREAT MODEL & PROBLEM SCOPE

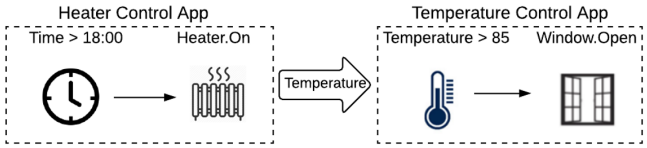


Figure 1: an Example of Inter-app Physical Interaction

One significant difference between IoT environments and conventional networks is that IoT devices have functions to interact with the surrounding physical environment, which makes it possible that IoT devices can interact with each other through *physical* channels even without network communications. However, those physical interactions cannot be seen directly from individual applications. As a result, an application, which has an impact on the physical environment, may unintendedly trigger another application to make unexpected reactions. Figure 1 shows an example of inter-app physical interaction, where a heater control application turns on a heater at a specific time, and a temperature control application [7, 39] opens windows when the temperature is higher than a pre-defined threshold. In this example, the *temperature* physical channel can connect the heater and the temperature sensor to create an inter-app interaction chain and lead to an unexpected action of opening windows.

Threat Model: In this paper, we focus on application-level IoT attacks on appified IoT platforms. Attackers attempt to misuse physical channels to trigger unexpected actions that may cause damages to the physical space. For example, a window opening action demonstrated in Figure 1 may cause a break-in. Since unexpected physical interactions exist among IoT applications, an attacker can launch an attack through either (1) *vulnerable applications*, which have design/implementation flaws that can be exploited by remote attackers or co-located malicious applications to escalate their privileges and cause security or safety issues, such as an unauthorized device control; or (2) *malicious applications*, which contain malicious program logic that can perform hidden behaviors [18]. We assume IoT devices are trustworthy. Hence, attacks targeting at manipulating device firmware vulnerabilities are not considered in this paper. We also assume that IoT platforms are trustworthy and uncompromised. Thus, we trust the APIs, communications, and management functions provided by IoT platforms.

Problem Scope: Since our design goal is to discover and analyze unexpected inter-app interactions on IoT platforms, attacks without exploiting inter-app interactions are not in our scope. For example, we do not investigate problems caused by devices or platform vulnerabilities [18]. Attacks targeting protocols flaws [34] and Denial-of-Service (DoS) behaviors [5] are also out of our scope. In addition, the problem of sensitive information leakage [1, 18] is also beyond the scope of this paper.

3 SYSTEM OVERVIEW

Our IoTMon system consists of three major components: i) *Application Analysis*; ii) *Interaction Chain Discovery*; and iii) *Risk Analysis & Mitigation*, as shown in Figure 2.

Application Analysis: This module includes two subcomponents, *Intra-app Analysis* (§4.1) and *Physical Channel Identification* (§4.2). The purpose of this module is to capture trigger-action control dependency of applications and discover physical channels that can link multiple intra-app interactions to form inter-app interaction chains. The intra-app interactions can be obtained through static program analysis. The physical channel identification aims at extracting channel related information from application descriptions, which are typically provided by application developers. These descriptions contain information about physical channels, such as *temperature*, *humidity*, *motion*, and *illumination*, which can be monitored or modified by the applications. In our design, we use NLP techniques to extract these channel information from application descriptions.

Interaction Chain Discovery (§4.3): This module takes all inter-app trigger-action interactions and physical channel information as input. The outputs are all possible inter-app interaction chains, which are generated by connecting intra-app interactions through proper physical channels.

Risk Analysis & Mitigation (§4.4): This module aims at providing a risk evaluation mechanism for inter-app interaction chains. First, our system models all interaction behaviors by mapping them into a high-dimensional space. In this space, we use intra-app interactions derived from official applications or verified third-party applications as the baseline of benign interactions to estimate risk levels of discovered inter-app interaction chains. Our risk evaluation mechanism calculates the distances between the inter-app interaction chains and the baseline to measure risks, where a large distance represents a high risk level. Based on risk levels of inter-app interaction chains, our system can then provide guidance to developers or users on risk mitigation.

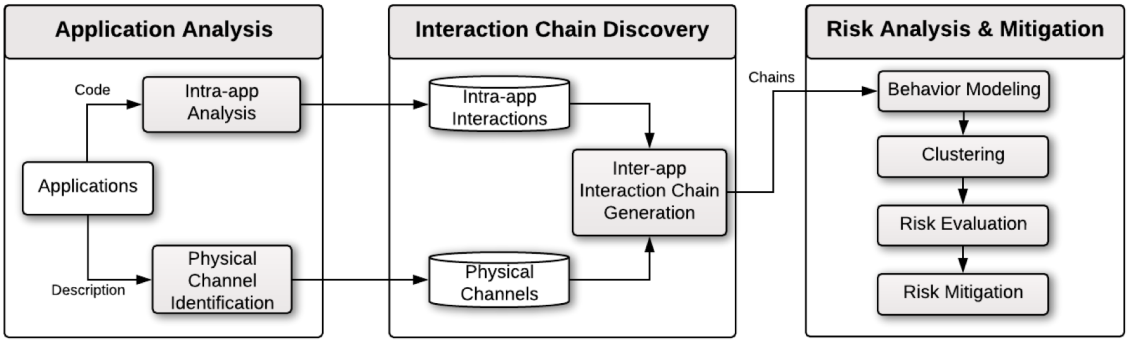


Figure 2: IoTMon System Overview

4 DESIGN AND IMPLEMENTATION

In this section, we present the detailed design and implementation of IoTMon. We first introduce our approaches for the intra-app analysis and the physical channel identification, respectively. Then, we discuss the procedure of inter-app interaction chain discovery. Finally, we describe our methods for risk analysis and mitigation.

4.1 Intra-app Analysis

4.1.1 General Policy Model. An IoT application's policies usually follow the "If-This-Then-That" (IFTTT) programing paradigm [30, 44]. Based on this observation, we propose a general policy model for our intra-app analysis. In IFTTT, "This" corresponds to the trigger capability and condition threshold. "That" represents the triggered action, such as changing a device's status. In IoT applications, we identify three important elements to describe the trigger-action relationship, and present a general policy model as shown in Listing 1.

Listing 1: A General Model for IoT Application Policies

```
1 <Name><Description>
2 <Trigger><Device:X><Condition>
3 <Action><Device:Y><Command>
```

We extract the following information from applications and map them into our general policy model.

(1) *Application description:* This part is typically located at the beginning of each application, which is provided by application developers.
(2) *Trigger condition and associated device:* Trigger conditions of an application are defined in the source code. For example, a trigger condition can be defined as "whether the temperature value is larger than a threshold".
(3) *Action and associated device:* Triggered actions are also defined in the source code. For example, a triggered action can be defined to turn a specific device on/off.

4.1.2 Intra-app Interaction Analysis. Our tool analyzes an application in three steps. First, an Abstract Syntax Tree (AST) is built for the application. Second, our tool analyzes the preference section in the code, where it claims all the capabilities and inputs of the application. The preference section is designed to let users setup proper devices and thresholds. Our tool traverses this section on AST and builds a list of capabilities and inputs. Third, our tool extracts triggers and actions of the application. Our tool identifies trigger conditions by parsing "subscribe" functions, which are defined for registering events on the platform to trigger actions. The actions can be identified by analyzing "installed" and "updated" functions. By tracing the control flows from subscribe functions to action functions, our tool extracts intra-app interactions in an application. We illustrate the detailed process of our static analysis through several examples in Appendix A.

4.2 Physical Channel Identification

Physical channels in an IoT environment are closely related to the physical interaction capabilities of IoT devices, *e.g.*, changing illuminance or increasing temperature. Several recent research efforts [31, 40] have demonstrated the possibility of extracting policy flows from application descriptions using NLP techniques. We observe that it is also possible to discover potential physical interactions of IoT devices through analyzing descriptions of an application. In our design, we leverage NLP techniques to identify physical channels from application descriptions.

We identify physical channels through three steps. First, we use NLP techniques to extract channel entity keywords from application descriptions. Then, we calculate similarities of extracted keywords by using Word2Vec [11] with a widely used language model (*i.e.*, Google News Vectors) [40]. Finally, based on the similarities of entity keywords, we cluster those entity keywords and identify physical channels based on entity keyword clusters. We next demonstrate the detailed process of our physical channel identification approach using an example application description: "Notify me when the humidity rises above or falls below the given threshold", which is from the *HumidityAlert* application in the Samsung SmartThings platform.

We first use the Stanford NLP tool [4] to parse this example description as shown in Figure 3. An application description usually contains information about its physical functions, *e.g.*, this example description in Figure 3 indicates that the application is related to *humidity*. After identifying entity keywords in an application

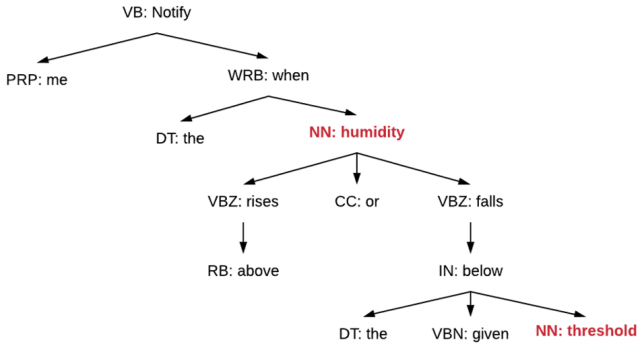


Figure 3: NLP for an application description: "Notify me when the humidity rises above or falls below the given threshold."

description, we next calculate similarities of these entity keywords. Then, we cluster similar entity keywords based on their similarities. For example, if there is an entity keyword "lights" mentioned in an application description, based on the keyword similarity, our system is able to cluster it with another similar keyword "bulbs". The channel identification is based on the aggregation (*i.e.*, the sum of similarity scores) of an entity keyword's similarity values within a cluster. The entity keyword with the highest aggregated value is considered as a representative keyword for the cluster. In the end, we check each cluster's representative keyword and remove non-physical-channel related keywords.

We apply our approach to analyze applications descriptions of official SmartThings applications and identify 217 entity keywords, which are then clustered into 16 different clusters. We finally identify 7 reasonable physical channels. The results are summarized in Table 1. For each identified physical channel, we give an example application, its description, and the number of keywords in its associated cluster.

4.3 Interaction Chain Discovery

Based on the intra-app interactions and physical channels identified in above steps, our system further discovers inter-app interaction chains.

4.3.1 System Channel Identification. In addition to physical channels, we observe that there are several system channels (in the Samsung SmartThings platform) that can be used to stitch different intra-app interactions. These system channels can be shared by multiple applications on the same platform. For the purpose of completeness, we also consider these shared system channels in our inter-app interaction chain discovery. During the intra-app interaction analysis, if a non-physical-channel capability is used as a trigger in one intra-app interaction and as an action in another one, we consider it as a shared system variable. We identify 4 such system channels and their related capabilities in the Samsung SmartThings platform, including *time*, *locationMode*, *switch*, and *lock*. Same as physical channels, we treat these system channels as connections between intra-app interactions in our analysis.

4.3.2 Inter-app Interaction Chain Discovery. We use a 2-element tuple (trigger, action) to represent the trigger-action *behavior* of an individual intra-app interaction. Algorithm 1 describes the procedure of discovering inter-app interaction chains. Let $AP_{tr,ac}$ denote all trigger-action behavior tuples in applications. Let $C_{ca,ch}$ store relationships between capabilities and channels, which can be obtained during the process of physical channel identification. Let S_s store all system channels. Algorithm 1 first reads all intra-app interactions from $AP_{tr,ac}$ as inputs. Then, it compares the channels used in each interaction to identify whether two intra-app interactions can be connected through the same channel. The outcome of the algorithm is a 5-element tuple, which contains a chain of inter-app interactions, i.e., (trigger1, action1, channel, trigger2, action2). Finally, our algorithm generates all potential inter-app interaction chains among applications.

Table 1: Physical Channels Identified from Official SmartThings Applications

Physical Channel	Example Applications	Descriptions	Number of Keywords
Temperature	Keep Me Cozy	"Changes your thermo--stat settings automa--tically in response to a mode change."	13
Humidity	Smart Humidity Vent	"When the humidity reaches a specified level, activate one or more vent fans."	10
Illumination	Brighten Dark Places	"Turn your lights on when a open/close sensor opens and the space is dark."	7
Location	Lock It When I Leave	"Locks a deadbolt or lever lock when a SmartSense Presence tag or smartphone leaves a location."	6
Motion	Light My Path	"Turn your lights on when motion is detected."	4
Smoke	Smart Home Monitor	"Monitor your home for intrusion, fire, carbon monoxide, leaks, and more."	4
Leakage	Flood Alert	"Get a push notifi--cation or text message when water is detected where it doesn't belong."	8

4.4 Risk Analysis & Mitigation

In Section 2, we show that attackers can exploit inter-app interaction chains to achieve malicious purposes. In order to measure potential risks of different inter-app interaction chains, we propose a risk evaluation method for quantifying inter-app interaction chains according to their influences on the physical space.

There are two challenges to determine whether an inter-app interaction chain is risky or not. First, we need a model to quantify physical influences incurred by different intra/inter-app interactions. Second, we need a baseline (*i.e.*, benign interactions) for the comparison with potentially risky interactions. To address these challenges, we introduce a *behavior modeling* method that assigns different physical channels with proper values, in order to calculate the distance between them. Official applications or third-party applications that have been verified/approved by platforms (*e.g.*, Samsung SmartThings) provide a good reference for benign interactions [13, 28]. Therefore, in our design, we use intra-app interactions

Algorithm 1: Algorithm for Interaction Chain Discovery

Input: $AP_{tr,ac}$, sets of intra-app interactions
$C_{ca,ch}$, sets of capabilities and their related physical channels.
S_{ca}, sets of capabilities and their related system channels.
Output: $INTAC$, sets of discovered interactions

```
1  foreach i ∈ AP_{tr,ac} do
2      foreach j ∈ AP_{tr,ac} do
3          if i ==j then
               /* Two intra-interactions are same */
4              continue
5          foreach k ∈ C do
6              foreach m ∈ C do
                   /* First identify capabilities of the
                      action and trigger */
                   /* Then check whether their related
                      channels are same */
7                  if i.ac == k.ca & j.tr == m.ca & k.ch ==
                    m.ch ∈ j then
                       /* Add a physical
                          interaction chain */
8                      INTAC ← {i, k.ch, j}
9                  foreach n ∈ S do
                       /* Same process for
                          system channels */
10                     if i.ac == n & j.tr == n then
11                         INTAC ← {i, n, j}
```

of trustworthy applications as the baseline to measure potential risks imposed by inter-app interaction chains. Our basic idea is that if an interaction is not in the baseline, it is likely a risky one. More specifically, trustworthy intra-app interactions are considered as safe interaction behaviors in our method. Then, we use the K-means [8] clustering to cluster all intra-app interactions to obtain the baseline. Finally, we are able to calculate risk scores of suspicious inter-app interaction chains based on the baseline. We further propose a method for risk mitigation, which can effectively reduce the number of risky inter-app interaction chains.

4.4.1 Behavior Modeling. For an intra-app interaction, we use the channel tuple to represent its trigger and action related channel information. Since an inter-app interaction chain involves multiple intra-app interactions and related channels, we use vectors to represent both inter-app and intra-app interaction behaviors. Each vector consists of all available physical/system channels, where each dimension/element in the vector corresponds to one channel, and the element's value represents the channel's status (*i.e.*, whether the channel is used, and whether it is used as a trigger or action). For instance, in our prototype implementation based on the Samsung SmartThings platform, we identify totally 7 physical channels, including *temperature, humidity, water, smoke, illumination, motion,* and *presence*, and 4 system channels, including *switch, lock, time*, and *locationMode*. In this case, we use an 11-dimensional vector to represent an interaction behavior instance.

The modeling process is summarized as follows and the first three steps are illustrated in Figure 4.

(a) *Channel Tuple Frequency Analysis*: We first extract intra-app trigger-action interactions and channel information from applications. To analyze the risk of interactions, we first map intra-app interactions to channel tuples based on the physical influences of their triggers and actions, respectively. We count the occurrence of a specific channel tuple out of all channel tuples as its frequency.
(b) *Channel Value Assignment*: Given the channel tuple frequency information, we assign values to different physical channels in a recursive manner, starting from the most frequently used channel. The difference in values reflects the correlations between physical channels. Note that the value assignment methods are different for physical and system channels.
(c) *Vector Value Assignment*: Based on the related channels and channels' values, all interactions can be mapped into a high-dimensional vector. Each dimension represents one channel, and the corresponding value represents its behavior (*i.e.*, either a trigger or an action on this channel).
(d) *Similarity Calculation*: The similarity between interactions is calculated based on the distance between corresponding vectors. We measure the risk levels of inter-app interactions by measuring the distances from them to the closest baseline cluster.

Our model captures the difference between channels in terms of the *frequency of their co-occurrence* in trustworthy intra-app interactions. For example, in our experiment, we observe that the temperature channel and the humidity channel more frequently appear together in the baseline interactions than the temperature channel and the motion channel. Therefore, according to our model, the difference between the temperature channel and the humidity channel should be smaller than the difference between the temperature channel and the motion channel.

Value Assignment to Physical Channels: We use the channel tuple (C_T, C_A) to represent each intra-app interaction, where C_T is the channel related to the trigger capability, and C_A denotes the channel related to the action capability. For example, considering a motionSensor capability as a trigger and a bulb switch capability as an action, the corresponding channel tuple is (motion, illuminance). To assign an initial value to each channel in an interaction behavior vector, we count the frequencies of all channel tuples in the baseline (*i.e.*, all trustworthy intra-app interactions) as illustrated in Figure 4 (a). The procedure of the value assignment starts from the channel with the highest frequency in all tuples, where we assign an initial value K (K can be an arbitrary value) to the first channel. Then, we assign a value to the next channel, which has the highest co-occurrence with the first assigned channel in all channel tuples. We define a *step length*, denoted by λ. The next value to be assigned is always increased by λ for each value assignment. For example, the second assigned value will be $K + \lambda$. This process is repeated until all channels have been assigned with values.

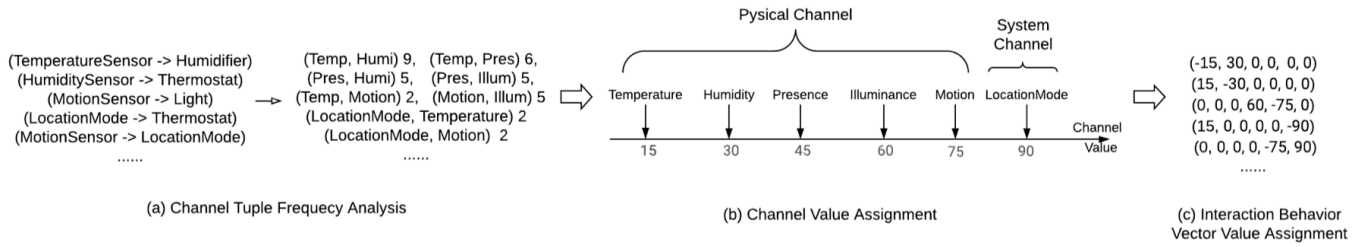


Figure 4: An Example of Behavior Modeling

Figure 4 (b) shows an example of channel value assignment with respect to 5 physical channels and 1 system channel. The number besides to each channel tuple in Figure 4 (a) indicates its overall frequency in intra-app interactions. In this example, the temperature channel has the highest frequency 17 (*i.e.*, it appears totally 17 times in all channel tuples). Thus, we choose it as the beginning channel and assign an initial value K (*e.g.*, K=15) to this channel. We assign the next value ($K+\lambda$ = 30) to the humidity channel, since it has the highest co-occurrence with the temperature channel (*i.e.*, appears with the temperature channel most frequently in all channel tuples). Similarly, the third iteration assigns the presence channel value to ($K+2\lambda$ = 45). By repeating this process, we assign values to all channels as shown in Figure 4 (b).

Value Assignment to System Channels: The physical impact of a system channel is hard to measure based on its co-occurrence frequency with a physical channel. A system channel may influence multiple devices, *e.g.*, the locationMode system channel can change the status of thermostats, lights, or heaters. Then, the status changes of these devices further influence their corresponding physical channels. As a result, a system channel may be indirectly related to multiple physical channels. In our design, for value assignment to a system channel, we consider all its related physical channels where its value is assigned to be the sum of all associated physical channels' values. For example, assume the locationMode influences temperature and motion channels. The temperature channel's value is 15, and the motion channel's value is 75. In this case, locationMode's value is assigned to 90.

Value Assignment to Interaction Behavior Vector: Given assigned values of individual channels, we are able to quantify interaction behavior vectors. Note that a channel can be either a trigger capability or action capability in a channel tuple (C_T, C_A). To distinguish them in a behavior vector, as long as a channel is associated with any *trigger* capability, we multiply a co-efficient "-1" with its channel value.

We illustrate our vector-based modeling approach using a 6-dimensional vector, including *temperature*, *humidity*, *presence*, *illuminance*, *motion*, and *locationMode*, as shown in Figure 4 (b). For the channel value assignment, assume the *temperature* channel is first assigned to 15, the *humidity* channel is assigned to 30, the *illuminance* channel is assigned to 60, and the *motion* is assigned to 75 (the step length is 15). The structure of the vector is shown as (temperature, humidity, presence, illuminance, motion, locationMode). Suppose there are three different interaction tuples (and each tuple corresponds to one vector) in the baseline benign interactions: A1 (temperatureSensor -> humidifier), A2 (humiditySensor, thermostat), A3 (motionSensor -> bulb). P1 represents an interaction that a temperature sensor detects temperature changes and then turns on a humidifier, which leads to changes in humidity. In this example, since the *temperature* channel is also used as trigger capability, the value of the *temperature* dimension is set to -10 in the vector value assignment. In Figure 4 (c), the vector value of A1 is (-15, 30, 0, 0, 0, 0), where "0" indicates that the channel is not involved in A1. Because the humidity in A2 is trigger condition, the vector of A2 is assigned as (15, -30, 0, 0, 0, 0). A3 indicates an interaction that a motion sensor detects a user' movement and then turns on a light. In this example, the value of *motion* dimension is assigned as -75, because it is a trigger condition. The vector value of A3 is assigned to (0, 0, 0, 60, -75, 0).

For inter-app interaction chains, we combine the values of intra-app interaction vectors. For the bridging channels between intra-app interactions, we treat them as a combination of trigger conditions, all of which multiply a negative coefficient in vector value assignment. We keep all the trigger conditions rather than only the first trigger and last action in the vector because bridging channels represent different paths of inter-app interaction chains. For example, assume we have two interaction chains C1 (motionSensor -> thermostat -> window.open) and C2 (motionSensor -> smokeSensor -> window.open). C1 and C2 have the same trigger and action, but medium channels are different, which results in different risk level for these chains. If we only keep the beginning trigger and final action for the inter-app interaction chain vectors, we could not distinguish those two different inter-app interactions' trigger paths.

Similarity Calculation: There are many existing approaches, such as Manhattan Distance [10], Minkowski Distance [9], and Euclidean Distance [6], using *distance* for similarity calculation. In our system, we also use distance to measure the similarity between vectors. The shorter the distance is between two vectors, the higher similarity is between the corresponding interactions. For simplicity, we use Manhattan Distance in our similarity calculation. Other distance metrics can be also applied to measure the similarity.

4.4.2 Risk Evaluation. We leverage the intra-app interactions of trustworthy applications as the baseline to evaluate the risk of an inter-app interaction chain. First, we cluster all baseline intra-app interactions by using the K-means algorithm [8]. The largest distance between each cluster's center and its boundary is considered as the cluster radius. If a testing interaction behavior vector does not belong to any known (trusted) cluster, we mark it as a risky

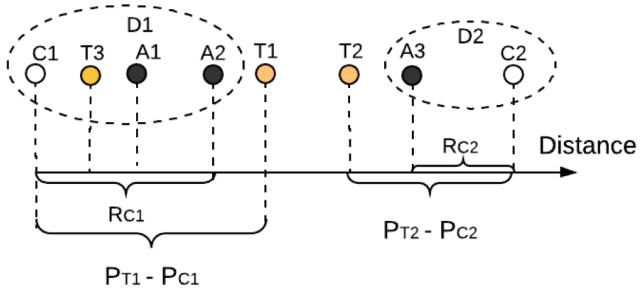


Figure 5: An Example of Risk Evaluation

interaction. In addition, the risk score is calculated based on the distance between the testing inter-app interaction and the closest trusted cluster's boundary.

Let D_i (i=1,2,...,n) denote the i_{th} cluster that contains a set of trustworthy intra-app interaction vectors in our baseline. We have n clusters in total. Let C_i (i=1,2,...,n) denote the center point of cluster D_i. T_i (i=1,2,...) is a vector of inter-app interaction chain to be tested (*i.e.*, testing vector). Our risk evaluation process is described as follows:

Baseline Generation: First, we use K-means to cluster all trustworthy applications' interactions. We set the number K empirically, which equals the sum of all channels. As shown in Figure 5, A1 and A2 are two trustworthy interactions in cluster D_1. The radius of cluster C_i is denoted by R_{Ci}. For example, the distance between C_1 and $A2$ is the radius of D_1 in Figure 5. Given a testing vector T_i, let $Position_{Ti}$ denote T_i's position in the high-dimensional feature space of our K-means clustering. $Position_{Ci}$ denotes the position of cluster D_i's center point in the high-dimensional space. If the distance between $Position_{Ti}$ and $Position_{Ci}$ is larger than R_{Ci}, we say T_i do not belong to D_i.

Risk Score: For a testing vector T_i, the risk score is calculated as follows:

$$RiskScore = \begin{cases} min\{|Position_{Ti} - Position_{Ci}| - R_{Ci}\},\ (i = 1, ..., n) \\ if\ T_i\ does\ not\ belong\ to\ any\ cluster \\ \\ 0,\ otherwise \end{cases} \quad (1)$$

If T_i belongs to any cluster D_i (i=1,...,n), the risk score is set to 0 (*i.e.*, normal case). Otherwise, it is considered as a risky case, and its risk score is set to $min\{|Position_{Ti} - Position_{Ci} - R_{Ci}|\}$, which denotes the closest distance between $Position_{Ti}$ and any cluster boundary in the baseline. We use the cluster boundary rather than the cluster center for calculating risk score. For example, in Figure 5, suppose T1, T2, and T3 are three testing vectors. The closest cluster center to T1 is C1, and the closest cluster center to T2 is C2. The risk score of T3 is zero, since it locates within cluster D1. The risk score of T1 is equal to the distance between T1 and A2, which is $(P_{T1} - P_{C1} - R_{C1})$. Because T2 is closer to the boundary of cluster D2, the risk score of T2 is $(P_{T2} - P_{C2} - R_{C2})$.

4.4.3 Risk Mitigation. We present a general risk mitigation method in IoTMon, which is flexible with respect to different scenarios. The key idea is to reduce the risks of unexpected inter-app interaction chains by *adding new trigger conditions*, *e.g.*, checking an additional related status. This can potentially prevent vulnerable applications from being maliciously triggered by unexpected applications. For application developers who can modify an application, IoTMon provides recommendations to guide them in reducing the risks of unexpected inter-app interactions. For normal users, IoTMon is able to give them risk warnings.

Revisiting the example in Figure 1, an enhanced application model is shown in Listing 2. The second trigger condition (highlighted with the gray background in Listing 2) can be added into the application. The condition means that a presence sensor must detect a person being at home before the temperature control application can open the window. Hence, the heater control application cannot directly open the windows without satisfying the added condition.

Listing 2: Window.Open Trigger Enhancement

```
1 <name: "Window Control">
2 <trigger1><device:temperature sensor 1><reportStatus:
      temperature.report > threshold >
3 <trigger2><device:location sensor 1><reportStatus:
4  sensor.detected>
5 If (trigger1) && (trigger2) == true Then
6 <action><device:Window><command:Open>
```

5 EVALUATION

In this section, we evaluate the effectiveness and efficiency of our IoTMon design from multiple aspects. We implement a proof-of-concept IoTMon system based on the Samsung SmartThings platform. We study totally 185 official SmartThings applications [17]. Our evaluation aims to answer the following questions:

- Whether all the 185 SmartThings applications follow the IFTTT programing paradigm? Can we always extract trigger-action relationships from them? Whether our application analysis tool can successfully extract trigger-action control dependency information and discover physical channels? (§5.1)
- How many inter-app interaction chains are found in our analysis? What is the most commonly used physical channel? (§5.2)
- Can IoTMon effectively detect high-risk interaction chains? In our interaction behavior modeling, we assign values to individual channels and interaction vectors. What are the impacts of different value assignment schemes on the risk evaluation? (§5.3)
- What is the performance overhead of IoTMon system? (§5.4)

5.1 Application Analysis

Intra-app Analysis: Among the total 185 SmartThings applications, we successfully extract trigger-action relationship information from 135 applications. For the rest 50 applications, 15 of them follow the IFTTT programing paradigm, but claim too many capabilities. Over-claimed capabilities would generate excessive intra-app interactions and make our risk analysis inaccurate. Therefore, we exclude these applications. Other 35 applications are either device drivers (*e.g.*, "Bose soundtouch connect", "Life360 (Connect)", and

Table 2: Examples of Intra-app Interactions

Applications	Triggers	Actions
Close the valve	waterSensor	valve.close
Its too cold	tempMeasure	switch.on
Keep me cozy ii	tempMeasure	thermostat.setCooling point
Whole house fan	tempMeasure	switch.on
Smart security	motionSensor	alarm.both

Table 3: Physical Channels and Associated Capabilities

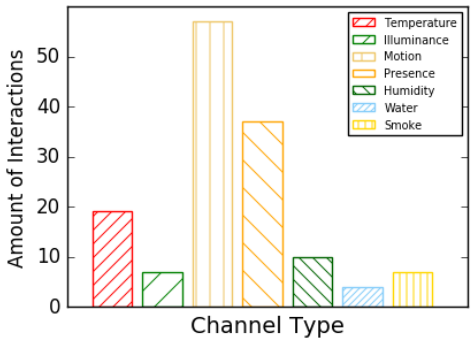
Channel	Capability
Temperature	temperatureMeasurement, thermostat, switch(AC)
Luminance	illuminanceMeasurement, switch(bulb), switchlevel
Motion	motionsensor, contactSensor, threeAxis
Humidity	relativeHumidityMeasurement, switch(vent)
Leakage	watersensor, valve
Location	location, presenceSensor
Smoke	carbonDioxide, smokeDetector

"Withings Manage") or service applications (*e.g.*, "Smart Energy Service" and "Severe Weather Alert") that do not contain physical interaction information. Table 2 illustrates 5 examples of intra-app interactions identified by our intra-app analysis, which can be described by the general policy model presented in Section 4.1. In this step, we finally identify 176 intra-app interactions in total.

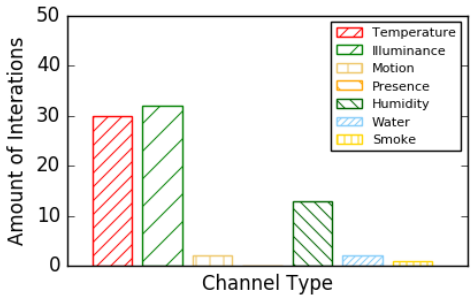
Physical Channel Analysis: In Table 3, we list all identified physical channels and their associated capabilities. We observe that some capabilities, such as *switch*, can be related to multiple physical channels. We classify these capabilities based on their usage descriptions. For example, if a switch's capability usage is described as "Turn on a light.", we derive that this switch is related to the illuminance channel. Figure 6 shows the amount of extracted intra-app interactions with respect to different physical channels. We report the trigger-related channels (in Figure 6(a)) and action-related channels (in Figure 6(b)), respectively.

From Figure 6, the motion channel is the most popular trigger conditions. However, since no device has the direct movement capability, it is also one of the least-used action capabilities. The temperature channel is used frequently in both triggers and actions. The luminance channel has more usage in actions, because a lot of motion conditions trigger the action of bulbs' switches.

Our physical channel analysis successfully connects 91.8% applications (124 out of 135 applications) to the correct channel categories. The failures mainly come from two cases. The first case is that an application description contains no physical channel related words. For example, the description of the application "IFTTT" says: "Put the Internet to work for you", which is incorrectly connected into the location channel in our analysis because of the "Internet". Another case is that an application description contains misleading words. For instance, in the application "coffee after shower", our method classifies this application into the humidity category because of the high similarity between "shower" and "humidity".



(a) Trigger-related Channel Analysis



(b) Action-related Channel Analysis

Figure 6: Physical Channels used in Intra-app Interaction Flows

5.2 Interaction Chain Discovery

Among 135 SmartThings applications, we observe that most of them have the capability to interact with the physical world. We generate all inter-app interaction chains following the Algorithm 1. According to Figure 6, the temperature is the most commonly used inter-mediate physical channel, which is involved in 570 inter-app interaction chains. However, most of these inter-app interaction chains are duplicated and the number of unique temperature-related chains is 25. After removing all duplicates, we generate 162 inter-app interaction chains in total.

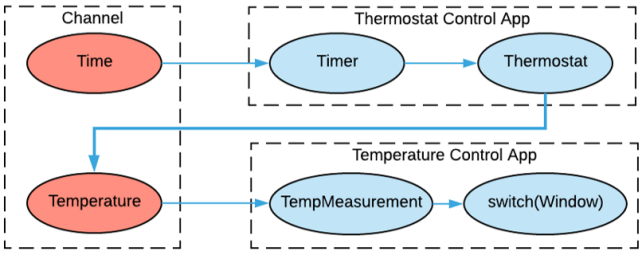


Figure 7: An Example of Inter-app Interaction Chain

Figure 7 shows an example of such interaction chains. The timer in *thermostat control application* triggers the action of a thermostat,

Table 4: Top 10 High-Risk Interaction Chains

No.	Trigger1 Capability	Action1 Capability	Potential Device	Channel	Trigger2 Capability	Action2 Capability	Potential Device	Risk Score
1	locationMode	switch	heater	smoke	carbonMonoxide	locationMode	alarm(window, lock)	70.75
2	time	switch	feeder, curtain, fan	motion	motionSensor	locationMode	bulb, window	64.81
3	time	locationMode	multiple devices	system	locationMode	lock	lock	62.92
4	locationMode	thermostat	thermostat	temperature	tempMeasure	switch	window, vent	62.75
5	time	switch	heater	temperature	tempMeasure	locationMode	multiple devices	60.01
6	switch	thermostat	thermostat	temperature	tempMeasure	locationMode	alarm(window, lock)	48.74
7	presenceSensor	switch	toaster	smoke	carbonMonoxide	locationMode	alarm(window, lock)	45.21
8	time	locationMode	multiple devices	system	locationMode	switch	heater	40.83
9	time	locationMode	multiple devices	system	locationMode	switch	bulb	40.50
10	watersensor	switch	bulb	illuminance	illuminMeasure	switch	bulb, heater	35.06

Table 5: Channel Similarity and Initialization

Channel Type	Target Channel	Level Distance	Assigned Value
Physical	Temperature	0	15
	Humidity	1	30
	Presence	2	45
	Illuminance	3	60
	Motion	4	75
	Leakage	5	90
	Smoke	6	105
System	Time	NA	105
	Switch	NA	120
	Lock	NA	150
	locationMode	NA	330

which triggers a switch in *temperature control application* since they share the same temperature channel. We present the complete results of our interaction chain discovery in Appendix B.

5.3 Risk Analysis

5.3.1 Channel Value Setting. We use the intra-app interactions extracted from official applications in the SmartThings platform as the baseline for risk evaluation. We use vectors to represent interaction behaviors where each vector consists of all available physical and system channels, as described in Section 4.4.1. To quantify interaction behaviors, we first assign values to different channels and subsequently initialize vector values. Table 5 shows the assigned values to different channels in our evaluation. We set the *temperature* as the beginning channel and set its value to 10, because overall it is the most frequently used channel in applications. For the shared system variables, we also list their values in this table. The value of a system variable is set as the sum of values of all associated channels. For example, the time channel is associated with the temperature, humidity, and illuminance channels, and is assigned 105.

5.3.2 High-risk Interaction Chain. We measure the risk score of a discovered inter-app interaction chain by comparing the distance between it and the baseline. Our risk calculation method identifies 48 risky interaction chains, and we examine our results manually and find 37 of them (77%) are truly risky, which create new and potential interactions that are not observed in the baseline (all 37 risky interaction chains are listed in Appendix C). To demonstrate the effectiveness of our value assignment, we also run an experiment 10 times with randomly initialized channel values. The random initialization method identifies more risky interaction chains. However, the amount of true positive risky interactions drops to 27 with a low positive rate of 34%.

Table 6: Risk Evaluation Result Assessment

Initialization Method	Total Interaction	Total Risky Interaction	True Positive	Positive Rate (%)
Our Method	162	48	37	**77**
Random Inil Method	162	79	27	**34**

The false positive results of our method mainly come from insufficient descriptions to the usage "switch" capability in applications. Since many devices can be connected to the switch capability, lacking such description can lead to over-connection, which creates non-existing and unpractical inter-app interactions. On the contrary, if the switch capability has an explicit explanation, our method could effectively evaluate its physical effect and connect them to the proper channel.

We list the top 10 high-risk interaction chains in Table 4. All the chains access indirect control of other devices through physical or system channels. Seven of them are new inter-app interactions, which create potential interaction chains through physical channels, *i.e.*, the 1st, 2nd, 4th - 7th, and 10th interaction chains. The remaining three create new interaction chains through system channels. Intuitively, if the first action and the second action are different in an interaction chain, it tends to have a higher risk score.

The 1st risky interaction chain links the location and lock through the smoke channel. There is no such direct interaction in the current official applications. We assume the switch is connected with a heater, which is able to generate smoke in a certain situation. If a heater is exploited to trigger the smoke alarm, it subsequently results in the action of windows opening and door unlocking. The 2nd risky interaction chain creates the new interaction from time to locationMode through the motion channel. The motionSensor can change the locationMode through the security monitor function. The time-scheduled motion should not trigger the change of locationMode. The 3rd chain is only based on the system variables,

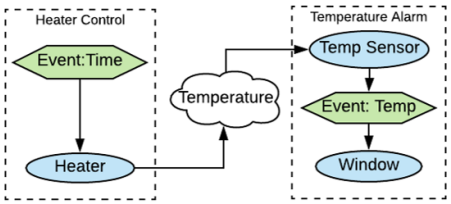


Figure 8: Two benign applications trigger an unexpected action through a physical channel.

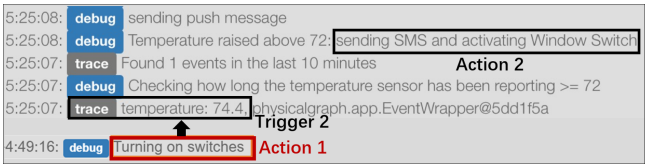


Figure 9: Logs on the SmartThings Platform in Scenario I

time and locationMode. Since the locationMode is a common component shared by all applications, the change of its status would trigger actions from multiple devices. The status of these devices may influence the status of locationMode in return. Hence, the time may trigger the unexpected changes to the home mode. The 4th chain unexpectedly triggers the switch's action via the temperature channel. The 5th chain is similar to the 2nd chain, where time can change the locationMode through the temperature channel. The 6th chain represents the situation that a switch is able to trigger a temperature alarm by utilizing a thermostat to influence the temperature channel. The 7th chain is triggered by a presence sensor that connects the toaster to the locationMode through the smoke channel, which can be used to trigger an alarm and subsequent actions of related devices, such as windows and locks. The 8th and 9th risky interaction chains are triggered by time channel. They are also able to trigger unexpected changes to the home mode. The 10th chain creates interactions between a water leakage and a switch through the illuminance channel. Since the switch can be connected to the bulb to send a leakage warning, and subsequently trigger actions of illuminance-related devices.

5.3.3 User Scenario. To demonstrate the real effect of potential inter-app interactions, we create two real-world scenarios based on our identified risky interaction chains.

Scenario I: Figure 8 shows a scenario where two benign applications (a heater control application and a temperature alarm application) share a temperature channel. Assume we use the temperature alarm application to monitor the temperature of an infant room. Once the heater control application turns on the heater, it leads to temperature raising. The increased temperature triggers the temperature alarm application to open the window, which may cause infant safety issues in this scenario.

Figure 9 shows the real logs on the SmartThings platform after the window control application has been triggered to open the window by a temperature raising event, which is caused by turning on a heater's switch for around 35 minutes.

Scenario II: As shown in Figure 10, the whole interaction structure in this scenario includes a malicious application (a device

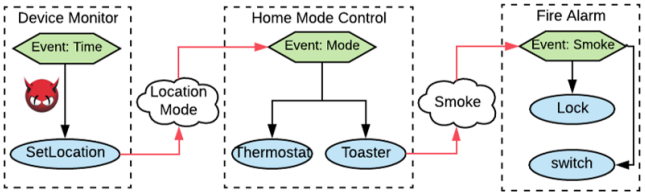


Figure 10: A malicious application triggers an inter-app interaction of two benign applications.

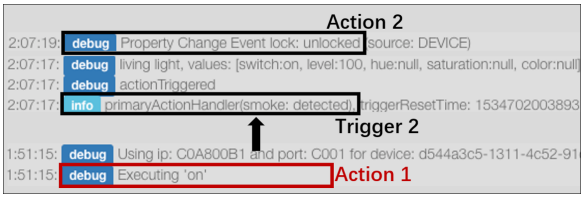


Figure 11: Logs on the SmartThings Platform in Scenario II

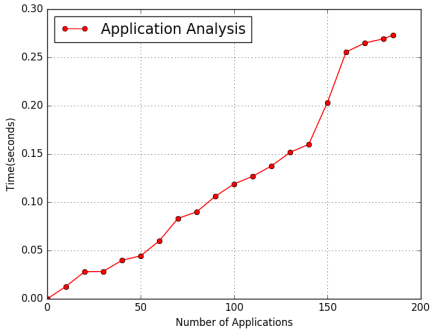
monitor application) and two benign applications (a home mode control application and a fire alarm application). The home mode control application changes the status of thermostats and toasters based on its mode status. The fire alarm application can unlock the door when any smoke is detected. The attacker uses the device monitor application, which is a third-party malicious application, and can pretend as a benign device status monitor to communicate with other applications stealthily.

The red arrows in Figure 10 illustrate the inter-app interactions, which could be utilized by the attacker. Assume the malicious application has been installed on the platform. It aims at modifying the status of *locationMode* to "Home" at a specific time stealthily by utilizing a system flaw [18]. Then, the *locationMode* triggers actions of other devices, which are defined in the home mode control application. We assume the home mode control application gets access to the thermostat and toaster. If a toaster is overheated, it may lead to the reaction of the fire alarm application, *e.g.*, unlocking the door, which makes intrusion possible. In this scenario, even a malicious application does not get access to a lock, it still can affect the lock's status indirectly.

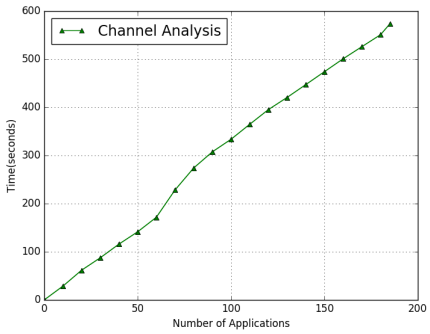
In this scenario, we use a First Alert smoke detector, a Z-Wave Schlage Lock, and a toaster controlled by a *wemo* outlet to implement a smart toaster. Figure 11 shows logs generated by the SmartThings platform after the fire alarm application has been triggered by a smoke event, which is caused by turning on a toaster switch for 16 minutes. Since the window control flow is similar to the benign case, we do not show the logs of that flow.

5.4 System Performance

In this section, we measure the performance of our system via processing all 185 official SmartThings applications. We test 20 times to calculate the average performance overhead on a desktop computer with an Intel 8700K CPU and 16 GB memories. The performance of the intra-app analysis is influenced by the number of applications and the complexity of each application. In our experiment, we measure the time of generating all the intra-app interactions. As shown



(a) Intra-app Analysis



(b) Channel Analysis

Figure 12: System Performance

in Figure 12a, the time for intra-app analysis of 185 applications is around 0.3s. The performance overhead of physical channel identification depends on the length of application description as shown in Figure 12b, which is around 573 seconds in total. The average time of risk analysis for 135 applications is approximately 1.2 seconds.

6 RELATED WORK

IoT Security: Existing research on IoT security has mainly focused on addressing *traditional* security issues on IoT environments, such as device or protocol flaws [16, 21, 36], malicious applications [18, 36], side channels [1, 24], and platform problems [18, 20, 27, 30].

Regarding devices flaws, some researchers focused on exploiting flaws on an IoT platform to attack the system. For example, Sivaraman *et al.* showed that an attacker could conduct a multi-step attack to compromise a local home network from the Internet by using a malicious mobile application [36]. By exploiting vulnerabilities in communication protocols, Ronen *et al.* developed a worm that could use the flaws in ZigBee to spread the worm among smart bulbs [34]. Chen *et al.* proposed a mechanism that could analyze device memory corruption vulnerabilities without analyzing devices' firmwares [16].

On application level, Jia *et al.* proposed a runtime authorization mechanism that uses context information to ensure the correctiveness of sensitive action execution [27]. Yuan *et al.* explored how to use static analysis and NLP to identify the inconsistency between the application's description and its functionality [40]. Moreover, Celik *et al.* proposed a static taint analysis tool called SAINT for tracing sensitive data flows in IoT applications [14]. They also introduced Soteria, a static analysis system for finding safety and security violations in an IoT application or IoT environment [15].

With respect to platform flaws, Fernandes *et al.* demonstrated that the overprivilege problem on the SmartThings Platform allows malicious applications to access non-authorized devices and sensitive event data [18]. In FlowFence, an information flow control and data isolation mechanism is proposed, focusing on solving sensitive information leakage on on the SmartThings platform [19]. Besides, Wang *et al.* have demonstrated that log information can be used to monitor malicious behaviors on the SmartThings platform [41].

Considering side channels on IoT platforms, Ronen *et al.* proved that attackers are capable of sending sensitive information by using strobed smart bulbs [1]. Recently, Han *et al.* demonstrated that identifying multiple physical impacts triggered by a specific event can help users to pair correlated devices, *e.g.*, human walking may change the status of motion, temperature and humidity sensors [24]. Although these research efforts revealed the possible influence of physical channels on smart home platforms, they mainly focused on finding physical impacts from a single physical event.

Although many IoT security problems have been addressed by existing research, our study revealed that a *new* type of security problem could happen due to specific physical functions provided by IoT devices. Especially, the ability of IoT devices to interact with physical surrounding needs to be monitored and controlled.

Risk Analysis: Many existing research focuses on the risk analysis of Android applications. Pandita *et al.* compared the results of NLP and static analysis to assess risks of Android applications [33]. They provided a mechanism to verify the consistency between an Android application's description and its behaviors. Some researchers utilized machine learning techniques to evaluate the risk of malicious Android applications. For example, Jing *et al.* used SVM to give the risk score of an application based on users' trusted applications [28]. Arp *et al.* used static analysis to extract features from applications and then used SVM to classify those features. These work focuses on application-level risk analysis and requires a large amount of training dataset, which does not exist currently on IoT platforms. In contract, our work focuses on evaluating risks of the physical interactions among applications on IoT platforms using limited intra-app interactions as a baseline.

7 DISCUSSION

In this section, we examine the limitations of our design and implementation, and discuss potential solutions for addressing those limitations.

Risk Analysis: Our risk analysis is sensitive to assigned channel values. The value assignment is based on the co-occurrence frequency of two channels in intra-app interactions. Intuitively, a higher frequency represents a stronger correlation between two channels. Hence, we give these channels similar values. However, such an assignment method may not be optimal. It would be an interesting problem to measure real-world correlations among different physical channels. In our future work, we plan to integrate other analysis methods, such as physical verification and machine

learning, into our risk analysis. We will also conduct user studies to verify the correctiveness of our risk analysis.

Risk Mitigation: Our risk mitigation method is relatively straightforward, which relies on developers to add more trigger conditions empirically. We plan to enhance our risk mitigation in two directions. First, it would be interesting to develop an automatic risk mitigation mechanism, which can change trigger conditions for risky interactions by learning existing applications' benign interactions. Second, we may develop a dynamic access control mechanism for IoT platforms without modifying application code. The access control policies can be generated automatically by machine learning techniques or from users' inputs. Thus, our system could achieve runtime interaction control without modifying existing applications.

Channel Identification: In our current design, we identify physical channels by clustering keywords from application descriptions. We may explore other methods to identify the existence of physical channels. For example, we may use system logs [41] or trace devices' sensor readings [24], to understand devices' interactions with different physical channels.

Description Integrity: Our system uses application descriptions to identify physical channels. We especially use the descriptions of official applications on the Samsung SmartThings platform for physical channel identification. If an attacker can craft malicious/misleading application descriptions in the applications, we need to verify the description integrity before using them in our system. We will investigate solutions to address such a problem.

8 CONCLUSIONS

In this paper, we have designed IoTMon, an IoT device physical interaction control system, which can discover all potential inter-app interaction chains and analyze risk levels of those interaction chains. We have implemented a prototype of IoTMon and evaluated it based on official SmartThings applications. Our evaluation results have demonstrated that IoTMon could effectively capture potential physical interactions among IoT applications and identify high-risk inter-app interaction chains.

ACKNOWLEDGMENTS

The authors thank Long Cheng for his help in polishing this paper. We also think Maxwell Harley for his help in the implementation of prototype system. This material is based upon work supported in part by the National Science Foundation (NSF) under Grant no. 1642143, 1723663, and 1700499. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF.

REFERENCES

[1] Extended Functionality Attacks on IoT Devices: The Case of Smart Lights, author=Ronen, Eyal and Shamir, Adi, booktitle=Proceedings of the 2016 IEEE European Symposium on Security and Privacy (EuroS&P), pages=3–12, year=2016, organization=IEEE.
[2] Zigbee. https://en.wikipedia.org/wiki/Zigbee.
[3] Home automation system market worth 79.57 billion usd by 2022. http://www.marketsandmarkets.com/PressReleases/home-automation-control-systems.asp, 2016.
[4] The stanford parser: A statistical parser. https://nlp.stanford.edu/software/lex-parser.html, 2016.
[5] Ddos attack that disrupted internet was largest of its kind in history, experts say. https://www.theguardian.com/technology/2016/oct/26/ddos-attack-dyn-mirai-botnet, 2017.
[6] Euclidean distance. https://en.wikipedia.org/wiki/Euclidean_distance, 2018.
[7] Fenestra. http://www.smartfenestra.com/products/, 2018.
[8] K-means clustering. https://en.wikipedia.org/wiki/K-means_clustering, 2018.
[9] Minkowski distance. https://en.wikipedia.org/wiki/Minkowski_distance, 2018.
[10] Taxicab geometry. https://en.wikipedia.org/wiki/Taxicab_geometry, 2018.
[11] Word2vec, doc2vec & glove: Neural word embeddings for natural language processing. https://deeplearning4j.org/word2vec.html, 2018.
[12] Apple. ios - home. http://www.apple.com/ios/home/.
[13] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens. DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. In *Proceedings of the 2014 Network and Distributed Security Symposium (NDSS)*, volume 14, pages 23–26, 2014.
[14] Z. B. Celik, L. Babun, A. K. Sikder, H. Aksu, G. Tan, P. McDaniel, and A. S. Uluagac. Sensitive Information Tracking in Commodity IoT. In *Proceedings of the 27th USENIX Security Symposium (USENIX Security)*, 2018.
[15] Z. B. Celik, P. McDaniel, and G. Tan. Soteria: Automated IoT Safety and Security Analysis. In *2018 USENIX Annual Technical Conference (USENIX ATC)*. USENIX Association, 2018.
[16] J. Chen, W. Diao, Q. Zhao, C. Zuo, Z. Lin, X. Wang, W. C. Lau, M. Sun, R. Yang, and K. Zhang. IOTFUZZER: Discovering Memory Corruptions in IoT Through App-based Fuzzing. In *Proceedings of the 22nd Network and Distributed Security Symposium (NDSS)*, 2018.
[17] S. Community. Samsung smartthing applications. https://github.com/SmartThingsCommunity/SmartThingsPublic, 2017.
[18] E. Fernandes, J. Jung, and A. Prakash. Security Analysis of Emerging Smart Home Applications. In *Proceedings of the 37th IEEE Symposium on Security and Privacy (S&P)*, May 2016.
[19] E. Fernandes, J. Paupore, A. Rahmati, D. Simionato, M. Conti, and A. Prakash. FlowFence: Practical Data Protection for Emerging IoT Application Frameworks. In *Proceedings of the 25th USENIX Security Symposium (USENIX Security)*, 2016.
[20] E. Fernandes, A. Rahmati, J. Jung, and A. Prakash. Decoupled-IFTTT: Constraining Privilege in Trigger-Action Platforms for the Internet of Things. *arXiv preprint arXiv:1707.00405*, 2017.
[21] D. Fisher. Pair of Bugs Open Honeywell Home Controllers Up to Easy Hacks. https://threatpost.com/pairof-bugs-open-honeywell-home-controllers-up-toeasy-hacks/113965/.
[22] B. Fouladi and S. Ghanoun. Honey, I'm home!!-Hacking Z-Wave Home Automation Systems. *Black Hat USA*, 2013.
[23] Google. Google home. https://madeby.google.com/home/.
[24] J. Han, A. J. Chung, M. K. Sinha, M. Harishankar, S. Pan, H. Y. Noh, P. Zhang, and P. Tague. Do You Feel What I Hear? Enabling Autonomous IoT Device Pairing Using Different Sensor Types. In *Proceedings of the 2018 IEEE Symposium on Security and Privacy (S&P)*, pages 678–694. IEEE, 2018.
[25] A. Hesseldahl. A hacker's-eye view of the internet of things. https://www.recode.net/2015/4/7/11561182/a-hackers-eye-view-of-the-internet-of-things/, 2015.
[26] IoTivity. Iotivity website. https://www.iotivity.org/.
[27] Y. J. Jia, Q. A. Chen, S. Wang, A. Rahmati, E. Fernandes, Z. M. Mao, and A. Prakash. ContexIoT: Towards Providing Contextual Integrity to Appified IoT Platforms. In *Proceedings of the 21st Network and Distributed Security Symposium (NDSS)*, February 2017.
[28] Y. Jing, G.-J. Ahn, Z. Zhao, and H. Hu. Riskmon: Continuous and Automated Risk Assessment of Mobile Applications. In *Proceedings of the 4th ACM Conference on Data and Application Security and Privacy*, pages 99–110. ACM, 2014.
[29] N. Lomas. Critical Flaw identified In ZigBee Smart Home Devices, 2015.
[30] C. Nandi and M. 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. ACM, 2016.
[31] K. Olejnik, I. Dacosta, J. S. Machado, K. Huguenin, M. E. Khan, and J.-P. Hubaux. Smarper: Context-aware and Automatic Runtime-permissions for Mobile Devices. In *Proceedings of the 2017 IEEE Symposium on Security and Privacy (S&P)*, pages 1058–1076. IEEE, 2017.
[32] OpenHAB. openhab - features - introduction. http://www.openhab.org/features/introduction.html.
[33] P. Rahul, X. Xiao, W. Yang, W. Enck, and T. Xie. WHYPER: Towards Automating Risk Assessment of Mobile Applications. In *Proceedings of the 22nd USENIX Security Symposium (USENIX Security)*. Citeseer, 2013.
[34] E. Ronen, A. Shamir, A.-O. Weingarten, and C. O'Flynn. IoT Goes Nuclear: Creating a ZigBee Chain Reaction. In *Proceedings of the 2017 IEEE Symposium on Security and Privacy (S&P)*, pages 195–212. IEEE, 2017.
[35] B. Schneier. The Internet of Things Is Wildly Insecure - And Often Unpatchable. *Schneier on Security*, 6, 2014.
[36] V. Sivaraman, D. Chan, D. Earl, and R. Boreli. Smart-Phones Attacking Smart-Homes. In *Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks (WiSec)*, pages 195–200. ACM, 2016.

[37] S. Smartthing. Smart home. intelligent living. https://www.smartthings.com/.
[38] S. Smartthing. Smartthings developer documentation. http://docs.smartthings.com/en/latest/.
[39] Steven. Windows automation in smart homes. https://smarthomegearguide.com/windows-automation-smart-homes/, 2018.
[40] Y. Tian, N. Zhang, Y.-H. Lin, X. Wang, B. Ur, X. Guo, and P. Tague. SmartAuth: User-Centered Authorization for the Internet of Things. In *Proceedings of the 26th USENIX Security Symposium (USENIX Security)*, pages 361–378, 2017.
[41] Q. Wang, W. U. Hassan, A. Bates, and C. Gunter. Fear and Logging in the Internet of Things. In *Proceedings of the 22nd Network and Distributed Security Symposium (NDSS)*, 2018.
[42] Wikipedia. Mirai (malware). https://en.wikipedia.org/wiki/Mirai_(malware).
[43] Wink. A simpler, smarter home. https://www.wink.com/.
[44] T. Yu, V. Sekar, S. Seshan, Y. Agarwal, and C. Xu. Handling a Trillion (unfixable) Flaws on a Billion Devices: Rethinking Network Security for the Internet-of-Things. In *Proceedings of the 14th ACM Workshop on Hot Topics in Networks (HotNets)*, page 5. ACM, 2015.

A INTRA-APP ANALYSIS

Our system can perform a lightweight static analysis on given IoT applications. It analyzes the rule grammar of applications by creating and parsing an AST of the application to find triggers and actions. The static analysis takes on four distinct phases with respect to the structure of a Samsung SmartThings application. The first phase converts the source code of the application into a Groovy AST. Since the SmartThings platform uses Groovy scripts instead of wrapping the source in a Groovy class, AstBuilder, provided by the Groovy project, cannot be utilized. Instead, the CompilationUnit class can be used to create the AST. The CompilationUnit class is a package provided by the Groovy language. It is the same one that the Groovy Console uses for parsing source code into an AST. An example of its usage is shown in Listing 3. The CompilationUnit class is suitable for semantic analysis, because it contains classes, imports, and variable scopes. Listing 3 demonstrates how to use the CompilationUnit class to generate the AST from a SmartThings application file, "source.groovy". By providing the source file and compilation unit option, the CompilationUnit class can create an ASTNode object that the analyzer can traverse to understand how triggers and actions are related to each other.

Listing 3: Using the CompilationUnit Class

```
def fileData = new File("./source.groovy")
CompilationUnit cu = new CompilationUnit()
cu.addSource(fileData)
cu.compile(Phases.SEMANTIC_ANALYSIS)
def ast = cu.getAst()
```

The second phase parses the preferences closure to make a list of inputs and capabilities. The preferences are shown to users as a menu where they can select what options their SmartThings applications should use. Since inputs of SmartThings applications typically have associated functions for changing the state of the capability, creating an easy-to-access list of these inputs makes the retrieval easy. The AST is easy to traverse if there are no "section" blocks or even those blocks do exist. Listing 4 provides an example of the preference, which showcases the intricacies of the preference closure. Specifically, it shows how sections are built from inputs, and inputs can even have sub-sections. One challenge with creating the list of inputs is that there are default system variables, such as "location" and "app", that SmartThings creates. The easiest way to fix the problem of undefined default variables is to manually add all default variables into the input list to allow them to be consumed later in the analysis process.

Listing 4: Preferences Closure

```
preferences {
  // Create section for Power Meter input
  section("Power Meter") {
      input "powerMeter", "capability.powerMeter"
  }
  // Create input section for contact
  input("recipients", "contact", title: "Send to") {
    input "sendPush", bool, title: "Send a push?", options:
        ["Yes", "No"]
    input "phone", "phone", title: "Send a Text?"
  }
}
```

The third phase maps inputs to outputs by parsing subscribe calls to get trigger handlers. The "installed" and "updated" functions tell SmartThings which previously-defined inputs are triggers and which are actions. Inside of the "installed" and "updated" functions, there are calls to "subscribe" and "schedule", which tell SmartThings what actions should be performed when the trigger is activated. The "schedule" function takes three arguments: a trigger name, a trigger channel, and a trigger handler. Listing 5 shows an example function for subscribing to events. It subscribes to all events, which are created and updated by powerMeter in the platform. By traversing the trigger handler function and checking for references to inputs located in the preference closure, the analyzer can tell which inputs are actions that the trigger calls. After performing three phases, our tool can have a list of triggers and their associated actions.

Listing 5: Installed Function

```
def installed() {
    // subscribe to powerMeter input and the
    // "power" attribute.
    subscribe(powerMeter, "power", handleMeter)
}
```

B INTERACTION CHAIN GRAPH OF 135 APPLICATIONS

Figure 13 shows a complete inter-app interaction chain graph of 135 Samsung SmartThings applications. Red nodes indicate physical/system channels. Blue nodes are triggers and actions from intra-app interactions. Blue edges between nodes represent intra-app interactions. All red edges are the paths of risky inter-app interaction chains.

C FULL LIST OF RISKY INTERACTION CHAINS

We provide a full list of risky interaction chains in Table 7. Note that "multiple devices" in potential device means the related device is not clear because there are too many potential related devices. In this case, we just choose one possible related capability each time to measure the risk of physical interaction of this capability. The potential devices are inferred from the applications' capability usage descriptions.

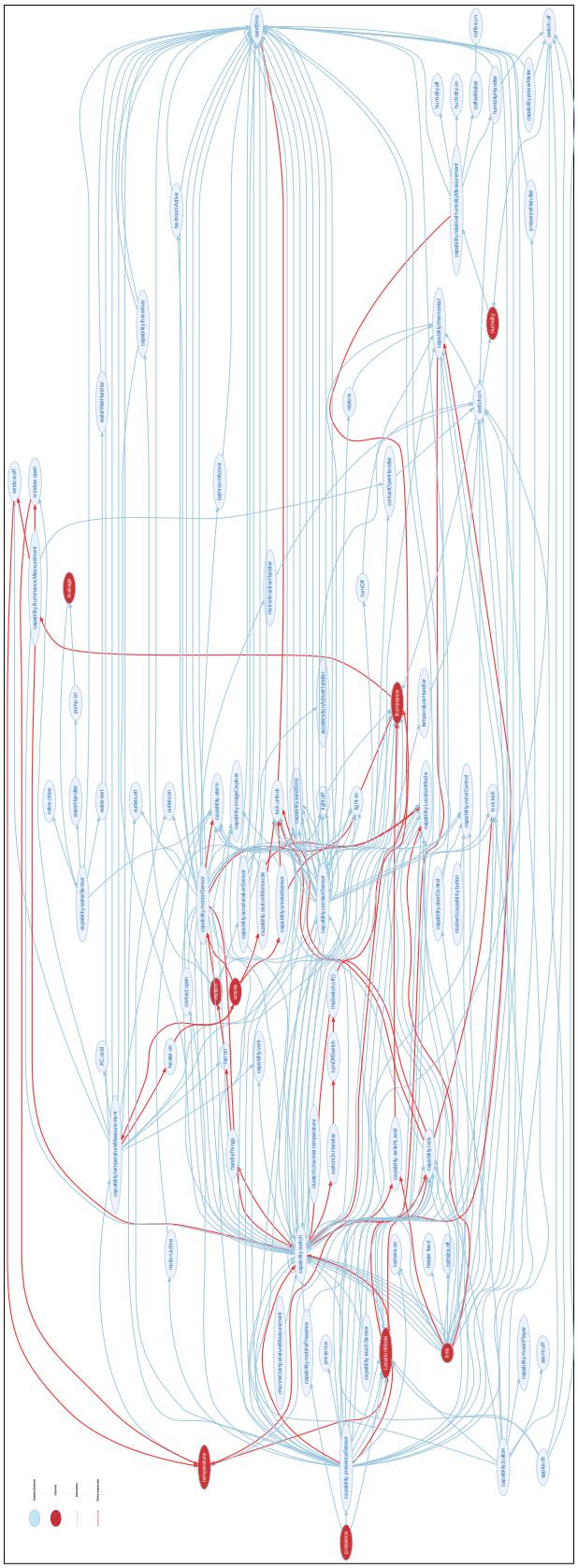


Figure 13: Inter-app Interaction Chain Graph of 135 SmartThings Applications

Table 7: Full List of Risky Interaction Chains

No.	Trigger1 Capability	Action1 Capability	Potential Device	Channel	Trigger2 Capability	Action2 Capability	Potential Device	Outlier Distance
1	locationMode	switch	heater	smoke	carbonMonoxide	locationMode	alarm(window, lock)	70.75
2	time	switch	feeder, curtain, fan	motion	motionSensor	locationMode	bulb, window	64.81
3	time	locationMode	multiple devices	system	locationMode	lock	lock	62.92
4	locationMode	thermostat	thermostat	temperature	tempMeasure	switch	window, vent	62.75
5	time	switch	heater	temperature	tempMeasure	locationMode	multiple devices	60.01
6	switch	thermostat	thermostat	temperature	tempMeasure	locationMode	alarm(window, lock)	48.74
7	presenceSensor	switch	toaster	smoke	carbonMonoxide	locationMode	alarm(window, lock)	45.21
8	time	locationMode	multiple devices	system	locationMode	switch	heater	40.83
9	time	locationMode	multiple devices	system	locationMode	switch	bulb	40.50
10	watersensor	switch	bulb	illuminance	illuminMeasure	switch	bulb, heater	35.06
11	waterSensor	locationMode	multiple devices	system	locationMode	lock	lock	34.91
12	locationMode	switch	heater, AC	temperature	tempMeasure	switch	heater, window	34.37
13	location	thermostat	thermostat	temperature	tempMeasure	switch	heater, window	34.12
14	motionSensor	switch	bulb	illuminance	illuminMeasure	switch	bulb, window, curtain	30.42
15	time	thermostat	AC	temperature	tempMeasure	switch	vent, heater	29.73
16	time	switch	feeder, fan, curtain	motion	motionSensor	illuminMeasure	window, bulb	27.06
17	time	locationMode	multiple devices	system	locationMode	thermostat	AC	26.87
18	carbonMonoxide	switch	alarm	motion	motionSensor	locationMode	lock, window	26.63
19	time	switch	bulb	illuminance	illuminMeasure	switch	curtain, window, bulb	26.48
20	motionSensor	locationMode	multiple devices	system	locationMode	lock	lock	25.29
21	waterSensor	switch	valve, bulb	illuminance	illuminMeasure	switch	curtain, window	22.64
22	time	switch	feeder, fan, curtain	motion	motionSensor	lock	lock	21.65
23	presenceSensor	switch	toaster	smoke	carbonMonoxide	locationMode	alarm(window, lock)	16.38
24	HumidityMeasure	switch	heater	smoke	carbonMonoxide	locationMode	alarm(window, lock)	16.15
25	timer	switch	feeder, fan, curtain	motion	motionSensor	switch	humidifier	16.11
26	waterSensor	locationMode	multiple devices	system	locationMode	switch	window, heater	16.02
27	waterSensor	locationMode	multiple devices	system	locationMode	switch	bulb	15.61
28	tempMeasure	switch	heater	smoke	carbonMonoxide	locationMode	alarm	12.96
29	time	switch	feeder, fan, curtain	motion	motionSensor	thermostat	thermostat	12.67
30	illuminMeasure	switch	feeder, fan, curtain	motion	motionSensor	lock	lock	11.15
31	time	switch	bulb	illuminance	illuminMeasure	switch	curtain, window, bulb	5.60
32	motionSensor	locationMode	multiple devices	system	locationMode	switch	window	4.65
33	locationMode	switch	bulb	illuminance	illuminMeasure	switch	curtain, window, bulb	4.21
34	time	switchlevel	bulb	illuminance	illuminMeasure	switch	curtain, window, bulb	4.06
35	waterSensor	locationMode	multiple devices	system	locationMode	switch	humidifier	3.11
36	time	switch	bulb	illuminance	illuminMeasure	thermostat	thermostat	1.74
37	motionSensor	switch	heater	temperature	tempMeasure	switch	heater	0.28