InteractionShield: Harnessing Event Relations for Interaction Threat Detection and Resolution in Smart Homes

Zhaohui Wang
The University of Kansas
zhwang@ku.edu

Bo Luo
The University of Kansas
bluo@ku.edu

Fengjun Li
The University of Kansas
fli@ku.edu

***Abstract*—The widespread adoption of IoT devices and applications in smart homes has transformed the way we engage with our living environments. While enabling seamless automation and intelligent functionalities, interactions between different IoT applications, typically through trigger-condition-action (TCA) rules, may introduce new interaction threats due to rule conflicts, which sometimes lead to severe security and safety risks. However, existing detection and defense approaches often tackle specific threat categories in isolation, thereby failing to deliver a holistic perspective and robust, comprehensive protection. In this paper, we present InteractionShield, a novel framework that systematically detects and resolves rule conflicts by leveraging a logic analysis model based on event relations. The InteractionShield framework formalizes event relationships, detects event interferences, and classifies rule conflicts. It generates risk scores and conflict rankings to enable comprehensive conflict detection and risk assessment. To address the identified interaction threats, an optimization-based approach is employed to mitigate risks while maintaining system functionality. Evaluated on large-scale real-world IoT datasets, InteractionShield effectively enhances system reliability, offering a robust solution for detecting and resolving rule conflicts in smart environments.**

***Index Terms*—Smart Homes, IoT, Security, Rule Conflicts**

1. Introduction

The Internet of Things (IoT) enables seamless device interactions and task automation through app-defined rules. However, increasing system complexity introduces significant security and privacy concerns [1]–[4], particularly when interactions among apps cause conflicts. Poorly coordinated rules may result in unintended behavior, reduced efficiency, or safety risks, known as cross-app interaction threats [5].

While existing approaches to detecting rule conflicts have made meaningful contributions [5]–[13] many tend to be ad hoc and narrowly focused, addressing specific scenarios rather than offering a unified and comprehensive framework. Without decomposing complex rules into atomic events, these methods fall short in capturing the full spectrum of event relationships or accounting for all possible interferences. As a result, their ability to detect conflicts remains *incomplete*.

Moreover, existing methods often define conflicts too broadly without integrating the usage context, which can lead to *inaccurate* detection. For instance, condition-interference threats [8] describe cases where one action triggers the condition of another rule, causing it to be evaluated as true. However, not all such interactions necessarily produce deterministic conflicts. In many cases, users deliberately configure rules to form chains of actions to achieve desired functionalities. Only when these interactions produce harmful or unintended outcomes should they be regarded as true conflicts. Additionally, some studies tend to over-generalize conflicts by combining basic conflict types into overly complex categories. This approach not only inflates the number of conflicts but also introduces unnecessary complications for their resolution.

To address these limitations, we present InteractionShield, a comprehensive framework designed to systematically and efficiently detect and resolve rule conflicts. InteractionShield begins by decomposing complex rules into atomic events, then formalizes their relations to identify all potential interferences among those events. From this foundation, it derives and categorizes every possible type of conflict. Building on this, InteractionShield evaluates the risk associated with each action and ranks conflicts according to the specific context and scenario in which they occur. The risk scores and conflict rankings not only quantify the severity of conflicts but also guide prioritization, enabling a targeted conflict resolution process. Finally, to maintain system functionality while reducing conflicts, InteractionShield uses an optimization-based strategy that preserves as many rules as possible while significantly reducing the overall conflict penalty. By combining exhaustive detection with context-sensitive risk assessment and optimization-based resolution, InteractionShield delivers a scalable and practical solution for managing rule conflicts in IoT environments.

The main contributions are summarized as follows:

- We design the InteractionShield framework, which decomposes rules into atomic events, identifies conflicts and interferences via event relations.
- InteractionShield integrates a risk-based approach for context-aware conflict ranking and an optimization strategy for balancing conflict resolution and applications' functionality.

- We implement a prototype of InteractionShield and demonstrate its scalability and effectiveness on large-scale real-world datasets.

2. Background and The Problem

2.1. Background

IoT Platforms and Apps. SmartThings, IFTTT, and OpenHAB are three prominent home automation platforms, each using a trigger-condition-action paradigm for managing IoT devices. SmartThings uses Groovy-based SmartApps, which rely on subscription or scheduling functions and capabilities (comprising commands and attributes). IFTTT applets and OpenHAB rules follow an if-this-then-that structure: IFTTT provides triggers, actions, and device endpoints in JSON, while OpenHAB adopts a Java-like DSL with things, items, and channels. Because they share a similar structure, apps from these platforms can be converted into a unified format, enabling rule extraction and interaction analysis.

App Interactions. As deployments scale, different apps increasingly affect other by sharing devices, physical channels, or data. One app may enable or disable events in another, or they may execute actions that conflict. These interactions can be intentional, such as chaining actions across multiple apps for advanced automation, or unintentional, potentially leading to conflicts or raising security and privacy concerns.

IoT Properties. In IoT systems, properties specify expected behavior to ensure correctness by avoiding undesirable states and achieving desired outcomes. We group them into two types: *rule properties*, which prevent rule conflicts (e.g., turning a heater on and off simultaneously), and *behavior properties*, which encode user-defined requirements (e.g., auto-locking a door when away). Violations may compromise safety, security, reliability, or overall system performance.

2.2. The Problem and Our Motivation

Over the years, a variety of terminologies and inconsistent definitions have been used to describe rule conflicts, leading to a fragmented understanding of the problem. To address this, we reviewed the literature on conflict detection published in the past ten years and made several key observations that highlight the ad hoc nature of prior approaches to detect IoT interaction conflicts. Building on these insights, we define 10 fundamental types of rule conflicts, as shown in Table 3, and reconcile the diverse terminologies used in earlier studies by mapping them to our definitions (see Table 11 and Table 15).

Observation 1: Prior work on IoT rule conflict detection lacks a systematic framework for identifying conflicts. Instead, they often rely on ad hoc, scenario-specific definitions that cover only a narrow subset of possible interactions. By defining just a limited set of conflict types, these approaches fail to capture many underlying event relationships and interference patterns. Furthermore, because they do not decompose events into atomic components, their analyses operate at a high level of abstraction and overlook subtle but critical rule interactions. As a result, most of existing conflict detection methods are inherently incomplete. For example, Alhanahnah et al. [9] defined an *Action-Action conflict* as two rules triggered by the same event but executing different actions, while Corno et al. [14] described *inconsistencies* as rules activated at (nearly) the same time attempting contradictory actions. Meanwhile, Ma et al. [15] introduced a *numeric device conflict* for scenarios where two actions impose different numeric settings on the same device that cannot be satisfied together. Similarly, Wang et al. [11] defined *action revert* and *action conflict* as cases of two actions with opposing effects executed in deterministic and nondeterministic order, respectively. Chi et al. [5] further refined the taxonomy with categories like *race condition* (the same trigger leads to actions that cancel each other out), *potential race condition* (two opposite actions occur regardless of a specific trigger), and *action revert* (a chain of two rules where the second rule's action undoes the first).

Despite differences in terminology and emphasis (e.g., whether rules share the same trigger, how closely timed the actions are, or whether mutual exclusivity is considered), all these definitions converge on the same core issue: two rules issuing contradictory actions. In other words, irrespective of whether the triggers are identical or not, a conflict arises whenever rules attempt to carry out contradictory actions simultaneously. By focusing only on high-level trigger-action relations via ad hoc categories, prior work overlooks fundamental event relations, such as logical dependencies between conditions and temporal constraints on rule execution, resulting in incomplete or inconsistent conflict detection.

Observation 2: Many IoT conflict-detection studies create seemingly new conflict categories by merging basic conflict types. This may obscure the analysis rather than clarifying it. These composite categories do not represent genuinely new conflict types, but rather combine patterns that could be captured by simpler definitions. This may lead to inconsistent and fragmented detection. For example, [6] introduced a conflict type called *tardy-channel-based rule blocking* using a scenario with three rules: (i) if the temperature falls below 10°C, turn on the heater; (ii) if the temperature exceeds 20°C, open the window; and (iii) if the temperature exceeds 26°C, turn off the heater. The interplay of these rules inherently yields multiple basic conflicts: rule 1 and rule 3 form a *chain loop*, i.e., the heater repeatedly turns on and off as the temperature oscillates; rule 1 and rule 2 create a *reduction impact conflict*, i.e., the open window counteracts the heater's effect; and rule 2 and rule 3 produce a *chain disable*, i.e., the open window prevents the temperature from rising enough to trigger the heater-off rule. The same study also defines categories such as condition dynamic blocking, scheduled condition blocking, and device disables, which are essentially variations of the chain disable pattern. All of these ostensibly new conflict types are composed of fundamental patterns rather than representing any genuinely novel category.

Observation 3: Previous studies often overlook the context-dependent nature of rule conflicts. Many approaches rely on rigid, arbitrarily defined conflict categories without considering the actual effects of the actions. This ignores that a

conflict arises only when a rule's action leads to an undesirable or incompatible outcome under specific conditions. For example, [8] introduced conflict types called *trigger-interference threats* and *condition-interference threats*, [5] proposed *condition enabling* and *chained execution*, and [9] defined *action trigger* and *action-condition match*. Despite differences in terminology, all these categories describe the same scenario: one rule's action serving as the trigger or condition for another. However, these works treat all such occurrences as conflicts, which overlooks the fact that users often intentionally chain rules to achieve specific goals. In such cases, rule chaining reflects purposeful design and should not be classified as a conflict.

Our observations indicate the absence of a systematic and comprehensive method for detecting conflicts and the need to resolve rule conflicts effectively. To tackle the challenges, we aim to answer three critical research questions: (1) *How to systematically identify and categorize all rule conflicts?* Existing research remains incomplete and fails to capture all fundamental event relations that lead to conflicts. (2) *How to measure the risk associated with actions and conflicts?* This requires a reliable method to assess action risks and rank conflicts. (3) *How can we best reduce rule conflicts while preserving the overall functionality of the system?* This requires defining a suitable objective function to optimize the system and minimize its risk score.

2.3. Threat Model

We assume that users may install multiple IoT apps from a variety of platforms and correctly associate them with their home devices. However, users often lack a comprehensive understanding of how these apps interact and may fail to recognize potential conflicts. These violations can originate from two primary sources: unintentional design flaws (e.g., poor logic between apps) or deliberate malicious intent (e.g., insertion of harmful code by an adversary to push the IoT environment into an insecure or unsafe state). In this work, we do not distinguish between the origins of these conflicting rules; instead, our focus is on detecting and mitigating the resulting violations.

3. Events and Conflicts

Different IoT platforms adopt varying paradigms, which can hinder comprehensive conflict detection. To address this, we seek clean logic with minimal abstractions. By merging components and breaking down complex rule structures into the smallest indivisible segments, we obtain a fine-grained representation that ensures a consistent level of detail across all IoT platforms. Conflicts are then identified through a modular, bottom-up hierarchy for precision and completeness.

3.1. Rules Reduction

Trigger-Condition-Action (TCA) rules are fundamental to event-driven programming and rule-based systems. These rules consist of three key components: a trigger (e.g., an incoming event or system state) that activates the rule, a condition that checks whether certain constraints are met, and an action that is executed only if both the trigger and condition are satisfied. Moreover, the condition itself can be interpreted as an additional trigger indicating a particular state, which allows us to merge the trigger and condition into a single logical unit, thereby simplifying TCA rules into Trigger-Action (TA) rules. We replace the separate trigger T and condition C with a new trigger defined as their conjunction $T \wedge C$, ensuring the action is executed only when both the original trigger and condition are satisfied.

3.2. Atomic Event

In an IoT environment, an atomic event is the most basic and indivisible component of an event condition. It represents an interaction between a connected device and either the physical environment or a user-issued command. An atomic event is defined as a single logical predicate comprising a device type, one of its attributes, and a constraint or target value applied to that attribute. For example, *sensor.temperature*> $70°F$ is an atomic event that represents the condition where the temperature measured by the sensor exceeds $70°F$. Each atomic event evaluates to either true or false based on the device's current state at a given moment.

Atomic events form the foundation for all IoT events, which are generally classified into two types: *trigger events* and *action events*. A trigger event represents a condition or change in state that initiates a system response. It is typically expressed in disjunctive normal form (DNF), which is a logical structure consisting of multiple conjunctions (ANDs) of atomic events combined using disjunctions (ORs). For instance, a trigger event "the door is closed and the lock is locked" can be represented as a conjunction of two atomic events: *door.door* == *closed* AND *lock.lock* == *locked*. In contrast, an action event specifies a command to be executed and generally consists of a single atomic event. For example, *light.on()* can be denoted by atomic event *light.switch* = *on*.

Let i be a device's unique identifier, d_i denote its device type (e.g., heater, lock), and l_i represent its physical location. Let $t = \langle t_b, t_e \rangle$ denote the time interval of an event, where t_b is the begin time and t_e is the end time. An atomic event is defined as $e_{i,c}^{t,\phi} = \langle i, c, t, \phi \rangle$, representing a basic, indivisible condition involving device i over the time interval time t. Here, $c \in \{T, A\}$ denotes the type of rule component, indicating whether the event is a trigger T or an action A, and ϕ is a Boolean predicate describing a device state or command (e.g., temperature> $70°F$). We adopt DNF to represent both trigger and action events. In DNF, a logical function f is defined as $f(\dots) = \bigvee_k \left(\bigwedge_j \psi_{k,j} \right)$, where each $\psi_{k,j}$ is an atomic event. Based on this formulation, a rule R is defined as $R = \langle E_T, E_A \rangle$, where $E_T = f\left(e_{i,T}^{t,\phi_i}, e_{i+1,T}^{t,\phi_{i+1}}, \dots, e_{m,T}^{t,\phi_m}\right)$ represents the trigger event, composed of a logical formula over $m - i + 1$ atomic events, $E_A = e_{j,A}^{t,\phi_j}$ represents the action event, typically consisting of a single atomic event.

TABLE 1: Event relations.

Groups	Event Relations	ID	Auxiliary	Description
Logical Relation	Logical Equal	L_{Equl}	$(d = d') \wedge (c = c') \wedge (\phi \equiv \phi')$	Same device/rule component, predicates logically equivalent.
	Logical Contain	L_{Cont}	$(d = d') \wedge (c = c') \wedge (\phi \subset \phi')$	Same device/rule component, one predicate proper subset of the other (not equal).
	Logical Intersection	L_{Ints}	$(d = d') \wedge (c = c') \wedge (\phi \| \phi') \wedge (\phi \cap \phi' \neq \varnothing)$	Same device/rule component, predicates have intersection (not equal/subset).
	Logical Disjoint	L_{Dsjt}	$(d = d') \wedge (c = c') \wedge (\phi \cap \phi' = \varnothing) \wedge (\phi \cup \phi' \neq \mathcal{U})$	Same device/rule component, predicates disjoint, union doesn't span entire domain.
	Logical Universal	L_{Univ}	$(d = d') \wedge (c = c') \wedge (\phi \cap \phi' = \varnothing) \wedge (\phi \cup \phi' = \mathcal{U})$	Same device/rule component, predicates non-overlap, union spans entire domain.
	Logical Independent	L_{Indp}	$d \neq d'$	Different device, same rule component.
Temporal Relation	Time Precede	M_{Prce}	$t_e < t'_b$	One event occurs entirely before the other event.
	Time Meet	M_{Meet}	$t_e = t'_b$	One event ends exactly as the other event begins.
	Time Overlap	M_{Ovrl}	$t_b < t'_b < t_e < t'_e$	One event starts before and ends during the other event.
	Time Contain	M_{Cont}	$(t_b < t'_b) \wedge (t_e > t'_e)$	One event completely contains the duration of the other event.
	Time Start	M_{Star}	$(t_b = t'_b) \wedge (t_e \neq t'_e)$	Both events begin at the same time but end differently.
	Time Finish	M_{Fnsh}	$(t_b \neq t'_b) \wedge (t_e = t'_e)$	Both events end at the same time but begin differently.
	Time Equal	M_{Equl}	$(t_b = t'_b) \wedge (t_e = t'_e)$	Both events have identical begin and end time.
Causal Relation	Causal Cause	S_{Caus}	$e_i \mapsto e_{i'}$	One event directly or indirectly generates or influences the other event.
	Causal Precondition	S_{Prcn}	$e_i \Rightarrow e_{i'}$	One event serves as a prerequisite for the other event to become possible.
Corefer Relation	Corefer Same	F_{Same}	$\Delta(h, l, l') > 0$	Both events influence the same physical channel and act in the same direction.
	Corefer Opposite	F_{Oppo}	$\Delta(h, l, l') < 0$	Both events influence the same physical channel but act in opposite directions.

Note: d denotes a device type and c a rule component; ϕ is a Boolean predicate; t_b and t_e are the begin and end times, respectively; and e_i is the event associated with device i. The function $\Delta(h, l, l')$ captures the influence of a shared channel h between devices at locations l and l'. We use $x \| y$ to indicate that neither is a subset of the other nor are they equal. The symbol $\mapsto$ denotes cause, and $\Rightarrow$ denotes a precondition.

TABLE 2: Event interferences.

Groups	Event Interference	ID	Logical						Temporal							Causal		Corefer		Description
			L_{Equl}	L_{Cont}	L_{Ints}	L_{Dsjt}	L_{Univ}	L_{Indp}	M_{Prce}	M_{Meet}	M_{Ovrl}	M_{Cont}	M_{Star}	M_{Fnsh}	M_{Equl}	S_{Caus}	S_{Prcn}	F_{Same}	F_{Oppo}	
Trigger Interference	Trigger Equal	TE	●								●	●	●	●	●					Triggers logically equal, occur simultaneously.
	Trigger Contain	TC		●							●	●	●	●	●					One trigger contains the other in same time period.
	Trigger Intersection	TI			●						●	●	●	●	●					Triggers partially overlap in logic and time.
	Trigger Disjoint	TD				●					●	●	●	●	●					Triggers are non-overlapping, do not form a universal.
	Trigger Universal	TU					●				●	●	●	●	●					Triggers form universal set, covering all conditions.
	Trigger Independent	TP						●	●	●	●	●	●	●	●					Triggers are independent of each other.
Action Interference	Action Equal	AE	●								●	●	●	●	●					Two actions logically equal, occur in same period.
	Action Mutex	AM				●					●	●	●	●	●					Two actions are logically disjoint in a short period.
	Action Opposite	AO					●				●	●	●	●	●					Two actions logical opposites, occur in same period.
	Action Accumulate	AA						●			●	●	●	●	●			●		Actions accumulate physical effects in shared space.
	Action Reduce	AR						●			●	●	●	●	●				●	Actions diminish physical effects in a nearby context.
	Action Force	AF						●			●	●	●	●	●		●	●		An action forces the execution of another.
	Action Block	AB						●			●	●	●	●	●		●		●	One action blocks or prevents another.
	Action Disorder	AD						●	●	●							●			Improper sequencing actions execute in wrong order.
Chain Interference	Cyber Enable	CE	●	●	●											●				Action activates trigger through cyber channel.
	Cyber Disable	CD				●	●									●				Action deactivates trigger via a cyber channel.
	Physical Enable	PE						●								●		●		Action activates trigger through a physical medium.
	Physical Disable	PD						●								●			●	Action deactivates trigger via a physical medium.

TABLE 3: Rule conflicts.

Rule Conflicts	ID	Trigger		Action	Chain	Auxiliary	Description	Ranking
		Deterministic	Conditional					
Duplication Action	**C.1**	TE,TC,TI TU†	TP	AE			The same actions are repeated within a short period or occurs across all situations.	Medium
Contradictory Action	**C.2**	TE,TC,TI	TP	AM,AO			The contradict actions are performed within a short period.	High
Accumulation Impact	**C.3**	TE,TC,TI	TP	AA			Different actions lead to physical effects accumulation.	High
Reduction Impact	**C.4**	TE,TC,TI	TP	AR			Different actions lead to the reduction of physical effects.	Low
Force Action	**C.5**	TE,TC,TI	TP	AF			Action A_i force the execution of action A_j.	Medium
Block Action	**C.6**	TE,TC,TI	TP	AB			Action A_i block the execution of action A_j.	Low
Disorder Action	**C.7**		TD, TU, TP	AD			Actions are executed in an incorrect or unintended order.	Low
Chain Enable	**C.8**‡	TE,TC,TI	TP		CE,PE§		R_i enable R_j through a cyber enable or a physical enable.	Medium
Chain Loop	**C.9**†	TE,TC,TI	TP		CE,PE§	$R_1 \circlearrowleft R_n$	Two or more rules form a circular loop.	Medium
Chain Disable	**C.10**‡	TE,TC,TI	TP		CD,PD§		R_i disable R_j through a cyber disable or a physical disable.	Low

Note: R represents a *rule*; $R_1 \circlearrowleft R_n$ denotes *loop* (i.e., $R_1 \rightarrow \cdots \rightarrow R_n \rightarrow R_1$); †denotes multiple-rule analysis, others are pairwise only; ‡indicates that a conflict is confirmed only if the action carries a medium or higher risk level; §indicates a possible deterministic conflict caused by physical influences.

3.3. Event Relation

There are four primary types of real-world event relations: *coreference*, *temporal*, *causal*, and *subevent*, which are identified in language models to capture the relationships between events in text [16]. These relations can be effectively applied to IoT systems. Both language models and IoT systems deal with events: what happened, when, and in what sequence. However, in IoT, an event is not only a small component of other events but can also have relationships such as equivalence, opposition, or other logical connections to other events. Therefore, we extend the concept of *subevent* relations to include broader *logical* relations.

Given two rules R_i and $R_{i'}$, each rule consists of a set of trigger events and action events. The complete set of events in a rule is denoted as $E(R) = E(T) \cup E(A)$, where $E(T)$ and $E(A)$ represent the trigger and action events, respectively. For any pair of atomic events $e_{i,c}^{t,\phi} \in E(R_i)$ and $e_{i',c'}^{t',\phi'} \in E(R_{i'})$, their relationship is defined by a 4-tuple:

$$\rho(e_{i,c}^{t,\phi}, e_{i',c'}^{t',\phi'}) = \langle L, M, S, F \rangle$$

Here, L denotes the logical relation, M represents the temporal relation, S captures the causal relation, and F

indicates coreference. We assess the logical relation between two atomic events by comparing their device types, rule components, and Boolean predicates. Temporal relations are determined by ordering their start and end timestamps according to Allen's interval algebra [17], which provides a complete set of interval-based relations. Causal relations capture how one event (the cause) contributes to another (the effect). In linguistics, a coreference relation links two or more expressions that denote the same entity. We adapt this idea to IoT systems by defining a coreference relation between two atomic events when they affect the same physical interaction channel. Since each device can be associated with one or more channels, and the effect of a shared channel may vary for devices in different locations, coreference must account for both channel identity and spatial context. In Table 1, we summarize 18 types of event relations with full descriptions and the equations for calculation.

3.4. Event Interference

Event interferences occur when interactions between events result in negative outcomes, such as functional inconsistencies or safety risks. These interferences are highly context-dependent, shaped by specific devices, scenarios, and the timing of occurrences. Interactions between events can lead to various types of interferences with differing severity. Consequently, the same set of events may function without conflict in one scenario but cause significant issues in another, emphasizing the importance of context-aware approaches to accurately identify and manage these interferences.

Let E_i and E_j each be either a trigger event E_T or an action event E_A. To capture how two events may interfere, we define an event interference function:

$$\tau\big(E_i, E_j\big) = \Big\{\langle L, M, S, F\rangle \in \mathcal{L} \times \mathcal{M} \times \mathcal{S} \times \mathcal{F}\Big\}$$

Here, $\mathcal{L}$, $\mathcal{M}$, $\mathcal{S}$, $\mathcal{F}$ respectively denote the sets of logical, temporal, causal, and coreference relations.

Building on the relations between trigger and action events, we distinguish three main relation types: *Trigger-Trigger*, *Action-Action*, and *Action-Trigger*. For each relation type, we account for both device functionality and usage context to systematically enumerate all possible interferences. To ensure coverage and completeness, we also draw on taxonomies from prior literature. This process yields three interference categories: *Trigger Interference*, *Action Interference*, and *Chain Interference*.

Trigger Interference arises when multiple triggers overlap, leading to redundancy or unintended coverage of all possible cases. Action Interference occurs when one action directly or indirectly alters, duplicates, or negates another action, producing undesirable outcomes. Chain Interference describes situations in which an action enables or disables another trigger whether via a cyber channel or a physical medium. Table 2 summarizes these categories and their defining relations. To detect an interference, at least one relation marked by black-dotted must be present in each dimension. Appendix B.1 provides a detailed explanation of how to calculate each interference.

3.5. Rule Conflicts

The primary sources of rule conflicts are action and chain interferences, with trigger interferences playing an auxiliary role. When a rule conflict always occurs whenever the rules are active, it is classified as *deterministic*; otherwise, it is classified as *conditional*. For any set of two or more rules, we compute each type of interference: trigger, action, and chain, then match these results against the entries in Table 3 to identify the specific conflict. We also check whether the rules form a loop to detect *Chain Loop* conflict. Each conflict identified in this way represents a violation of *rule properties*.

Duplication Action conflict occurs when multiple rules trigger the same action within a short period of time, leading to inefficiencies, resource wastage, or harmful cumulative effects. For instance, in healthcare, duplicate actions could result in medication overdoses, posing serious health risks. Similarly, in agriculture, duplicate actions in irrigation systems might cause overwatering, leading to soil degradation, or excessive fertilizer application, which could harm crops.

Contradictory Action conflict arises when two or more actions are inherently incompatible, such as being opposite or mutually exclusive, making coexistence impossible. For example, setting a thermostat to both 70°F and 80°F simultaneously is not feasible; turning a light on and off at the same time results in a conflict.

Accumulation Impact conflict arises when multiple actions have a cumulative effect that amplifies the physical impact on the same environment over time, leading to unintended outcomes. For example, activating both a heater and a fireplace simultaneously may cause the temperature to rise quickly, leading to overheating or even a fire hazard.

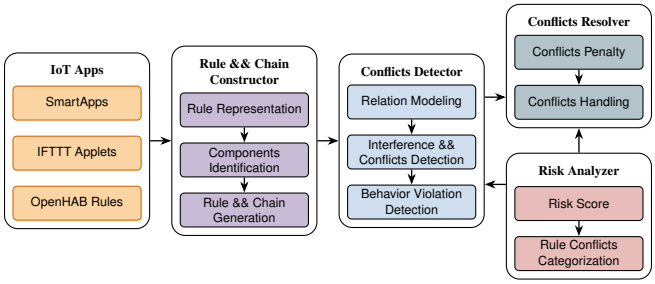
Reduction Impact conflict occurs when two or more actions interact in a way that diminishes the overall physical impact on the same environment. For example, in winter, if one rule activates a heater and another opens a window, the combined effect may reduce the room's ability to heat effectively, leading to wasted energy and inefficiency.

Force Action conflict occurs when one action forces the execution of another, leading to unintended outcomes. For example, a sprinkler operates through a valve. If the valve switches from off to on while the sprinkler is already on, it will be forced to spray water regardless of the intended state or conditions, potentially leading to unintended water usage.

Block Action conflict occurs when one action prevents or blocks the execution of another action, typically due to causal or physical constraints. For example, in a lighting system, if one rule cuts power by turning off the outlet, another rule attempting to turn the light on will be blocked because the power is no longer available; in a security system, if one rule locks a door, another rule trying to open the door may be blocked by the lock mechanism.

Disorder Action conflict refers to a situation where actions are performed out of the expected or correct order, disrupting the intended flow or process and often causing inefficiency or malfunction. For example, in an irrigation system, if the system is set to water plants before the soil moisture level is checked, it could lead to over-watering or ineffective

Figure 1: Architecture of InteractionShield.



Listing 1: The rule representation format.

```
predicate ::= (attribute)(operator)(value)          1
event     ::= (device).(predicate)                  2
trigger   ::= (event)+                              3
action    ::= (event)                               4
rule      ::= (trigger)+ ⇒ (action)                 5
```

watering; in a security system, if a rule to lock a door is executed before the door has fully closed, the locking action may fail or cause damage.

Chain Enable conflict occurs when one rule activates another, resulting in an unintended or conflicting outcome. This type of conflict occurs when the action executed by the second rule interacts with the system in a way that leads to unsafe consequences. For instance, if the second rule turns on a faucet, potentially causing a flood, a chain enable conflict is triggered, creating a hazardous condition.

Chain Loop conflict arises when a sequence of rules triggers a cyclical chain of actions, leading to a continuous, repetitive loop. This type of conflict can cause inefficiencies, system malfunctions, or unsafe conditions, as the ongoing loop may drain resources and overload the system.

Chain Disable conflict occurs when the activation of one rule disables another rule. This type of conflict arises when the disabled rule is crucial for maintaining safety or proper functioning, and its deactivation leaves the system vulnerable to failure. For example, if the second rule is to trigger the fire alarm, but it is disabled by the first rule, then in the event of a fire, it puts the user at significant risk.

4. InteractionShield Design

In this section, we present InteractionShield, a conflict analysis framework developed to detect and resolve conflicts in a multi-app, multi-platform environment. As illustrated in Figure 1, InteractionShield analyzes the app, constructs rules and chains (i.e., *Rule and Chain Constructor*), evaluates risk scores and classifies rule conflicts (i.e., *Risk Analyzer*), detects conflicts (i.e., *Conflicts Detector*), and resolves conflicts (i.e., *Conflicts Resolver*).

4.1. Rule and Chain Constructor

Rule Representation. Despite using different programming languages, IoT apps across platforms share common structures that support rule extraction. The process involves identifying trigger and action components, then analyzing the rule models implemented in each app. Rules are then extracted based on the representation format provided in Listing 1. An IoT app may contain multiple rules, each consisting of one or more triggers and a corresponding action. Both triggers and actions are expressed using atomic events, which serve as the fundamental units of rule representation. An atomic event is defined by a device type (e.g., *light*) and a binary predicate. A binary predicate consists of an attribute (e.g., *switch*), an operator (e.g., =), and a value (e.g., *on*).

Components Identification. Each IFTTT applet can be crawled as a key-value mapping, from which we can directly identify all rule components. SmartApps are written in the Groovy, while OpenHAB apps use a Java-based domain-specific language (DSL) with Groovy-compatible syntax. To analyze both, we parse their source code into abstract syntax trees (ASTs) using the Groovy compiler's *ScriptToTreeNodeAdapter* class and then build a control flow graph (CFG) for static analysis.

First, we extract device variable names and capabilities from the *input* method call nodes (in SmartApps) or identify all properties from *items* definitions (in OpenHAB), and the user assigns each variable its device type. Next, we traverse the CFG to locate action nodes, which appear as method-call nodes comprising a receiver, a method name, and an argument list (e.g., *thermo.auto()* or *light1.sendCommand(ON)*). If the receiver matches a known device variable and the method name corresponds to an officially documented command, we mark that node as an action. We normalize these commands into binary predicates, for instance, converting *light.on()* into *light.switch* = *on*. Next, we identify the entry nodes where the triggers are located. In SmartApps, the *subscribe* method or calls to *schedule* functions serve as potential entry nodes. They typically appear within lifecycle methods such as *initialize*, *install*, or *update*. The *subscribe* method binds event handlers to device attributes, locations, or app touch. Scheduling functions (i.e., *schedule* or *run**) trigger event handler methods at specific times. OpenHAB rules follow the format "when *trigger*, if *condition*, then *action*", entry points are defined by the clause between *when* and *then*.

Rule and Chain Generation. We traverse every execution path from entry point to action node, recording all predicates encountered along the way. From the Boolean expressions in *if* statements and *case* clauses within *switch* statements, we derive constraints for each path. These entry points and associated constraints are used to generate trigger events, while the action nodes are converted to action events. Next, for each app, we extract its complete set of rules, for each rule, identify all adjacent rules to which it may link. Finally, we apply a depth-first traversal to enumerate every possible rule chain, extending each chain to its maximal length.

4.2. Risk Analyzer

In Section 3.5, we analyzed and categorized rule conflicts. Next, we present the evaluation of risk scores. Actions directly affect the environment and change device states,

making them the primary source of potential risks. Triggers, in contrast, serve only as conditions or events that initiate actions and do not alter the system state themselves. Consequently, a rule's risk score is determined by the risks associated with its actions. At the chain level, the overall risk score is defined by the final rule in the chain, since its action ultimately determines the risk of the entire sequence.

The risk associated with a device in an IoT system is determined by its type, state, and operating scenario. For instance, a locked door during a fire emergency poses a serious safety hazard by blocking evacuation routes. Conversely, an unlocked door in an empty home compromises security, raising the risk of intrusion. To accurately evaluate this risk, we classify each device's operating scenario into one of three categories: *Safety*, *Privacy*, and *Convenience*. Devices that prevent or mitigate physical harm or emergencies belong to the *Safety* category. Devices that could expose sensitive information fall under *Privacy*. Finally, devices mainly used for ease of living, without significant safety or privacy implications are classified as *Convenience*. We analyzed a wide array of common IoT devices and grouped them into clusters (see Table 13). We then examined each cluster's possible states and assigned a risk score for every scenario, as shown in Table 14. For simplification and consistency, we normalize all device states to just two values: *on* and *off*. For example, a door that is *open* maps to *on*, while one that is *closed* maps to *off*. This approach standardizes risk evaluation across various devices and scenarios. When an IoT app's source code explicitly includes a category field (e.g., *Safety & Security*, *Convenience*), we use it to determine the app's usage scenario. If this field is absent, we assign the scenario that yields the highest risk score for that action. Once an app's scenario has been determined, we then consult Table 14 to obtain the action's corresponding risk score.

4.3. Conflicts Detector

To detect rule conflicts across multiple apps, we begin by modeling the logical, temporal, causal, and coreference relations between events (Table 1). Next, we analyze how each relation gives rise to trigger, action and chain interference, as detailed in Table 2. Finally, we apply the procedure summarized in Table 3 to pinpoint the resulting rule conflicts.

Relation Modeling. We describe how to model each relation using rule components, timing, predicates, device types, and device locations, ensuring conflicts are detected accurately.

Logical Relation. For any atomic events, we begin by identifying their device types and determining whether they serve as triggers or actions. We then extract the corresponding Boolean predicates. If the atomic events involve different device types, they are considered logically independent. To analyze other logical relations, we use the Python library SymPy, which supports symbolic mathematics. This allows us to determine whether the predicates are equal, contain one another, intersect, are disjoint, or together form a universal set. Each predicate is translated into either a SymPy interval or a finite set, and we apply SymPy's built-in methods to classify their relationships. When both predicates involve known constant values, the results can be computed directly. However, when the predicates include unknown variables, we enumerate all possible relationships between those variables to derive all valid outcomes.

TABLE 4: Dependency channels and device dependencies.

Channel	Dependable Devices	Dependent Devices
Electricity	outlet, plug	blender, boiler, cooker, cooktop, dishwasher, fireplace, freezer, fryer, kettle, light, microwave, oven, printer, projector, TV, cleaner, fan, dryer, mop, refrigerator, vacuum, washer
Water	water, valve, faucet	dishwasher, kettle, sprinkler, sprayer, washer
Gas	gas	cooktop, fireplace, heater, vehicle, oven, stove

Temporal Relation. While fully accurate time modeling cannot be achieved through static analysis, we assume that the temporal relations between events satisfy the timing constraints specified in Table 2. Transmission delays may cause events to arrive out of order, potentially altering the intended execution sequence of rules that rely on concurrent, consecutive, or adjacent events. Such cases represent execution-level conflicts rather than contradictions in temporal logic and are rare in practice. Therefore, we assume no transmission delays and that an action executes immediately once its trigger fires. This assumption enables conservative static analysis to capture a broader range of event interferences and detect more potential conflicts. Furthermore, the logical relations between triggers help determine the temporal relations between actions: independent triggers yield nondeterministic execution, overlapping triggers lead to simultaneous actions, and disjoint triggers preclude concurrent execution.

Causal Relation. The *Cause* relation examines whether one event can influence another directly or indirectly, either through cyber or physical channels. For cyber channels, we compare the device types and states associated with both atomic events. If they are the same, a *Cause* relation is inferred. For physical channels, we refer to the device associations and channel mappings described in [18]. If both events involve devices that share the same physical channel, a *Cause* relation is established. The *Precondition* relation explores dependencies between dependable and the devices that rely on them (i.e., dependent devices). Dependent devices cannot function if their corresponding dependable devices are unavailable (e.g., turned off). We have identified three primary dependency channels: *Electricity*, *Water* and *Gas*, which are summarized in Table 4, along with the dependable devices and the corresponding dependent devices. Certain devices (e.g., ovens) may depend on multiple channels (e.g., electricity and gas). An event is considered the *Precondition* of another if the device associated with the first event appears in the set of dependable devices, and the device associated with the second event appears in the corresponding set of dependent devices.

Coreference Relation. The physical impact varies depending on the type of device and its current state. Table 5 categorizes devices according to their associated physical channels and effects. Devices are grouped into three types based on their effects: devices of type 1 or type 2 produce the same physical effects when activated, while those of type 1 and type 2 have opposing effects. Devices of type 3 exhibit variable effects

TABLE 5: Different types of devices grouped by physical channels.

Channel	Type 1	Type 2	Type 3
Humidity	dehumidifier, fan, vent	humidifier	
Luminance	light, blind, curtain		
Power	A/C, cooler, fireplace, heater, kettle, stove, oven		
Smoke	fireplace, heater, stove, oven	door, purifier, window	
Sound	alarm, player, speaker, TV		
Temperature	fireplace, heater, stove, oven	cooler, fan	A/C, thermostat, door, window
Water	faucet, sprayer, sprinkler, valve		

depending on their states, for instance, an air conditioner may operate in heating or cooling mode based on its settings.

However, Coreference relations such as *action accumulate*, *action reduce*, and *physical enable* and *physical disable* depend strongly on real-world environmental factors (e.g., location, temperature, and humidity). Consequently, an environmental model is required to accurately represent how IoT devices interact in actual usage scenarios.

Building on the research from IoTSeer [19], we have adopted physical flow functions across 6 channels: *Temperature*, *Humidity*, *Illuminance*, *Sound*, *Motion*, and *Smoke*. Additionally, we have introduced a flow function for the *Power* channel to further enhance the model. Each device is assigned a power consumption value, allowing us to either sum the usage of all active devices to assess whether it exceeds the power threshold, or to compute cumulative energy consumption over time and compare it against predefined energy limits.

We apply a real-world environmental model to compute $\Delta(h, l, l')$, which captures the directional influence of the shared channel h between devices located at l and l', based on the specifications in Table 5. A positive result signifies a *Same* relation, whereas a negative result indicates an *Opposite* relation. To enhance usability, a user interface (Figure 6) is provided that allows users to specify the locations of IoT devices and initialize parameters such as temperature and humidity. Coreference relations are evaluated by determining whether there exists any point in time at which the relation holds. If such a moment exists, the coreference relation is deemed valid.

Interference and Conflict Detection. For each set of installed apps, we first detect *trigger*, *action*, and *chain* interferences, as described in Section 3.4. These interferences are then used to classify rule conflicts according to the criteria in Table 3. To distinguish between deterministic and conditional conflicts, we first examine the trigger interferences listed in Table 3. For chain-related conflicts, the analysis is refined further: if the chain is formed via cyber channels, trigger interferences are used to determine the likelihood of conflict; for chains established through physical channels, the conflict is deterministic as long as the physical connection is valid, regardless of trigger interferences.

Conflicts **C.1** and **C.9** require analyzing interactions among multiple rules, whereas the remaining conflict types involve only pairwise rule analysis. For conflicts **C.8** and **C.10**, we evaluate the action's risk score against a predefined risk level (set to medium by default in this paper). A conflict is confirmed only if the action's risk score exceeds this threshold, otherwise it is dismissed. For conflict **C.7**, users may specify additional action-ordering constraints based on their preferences (e.g., *unlock the door* should occur before *door opens*). In the case of **C.9**, we apply a cycle detection algorithm to identify loops in chains and assess whether a conflict exists. If the *Chain Enable* conflict between every pair of adjacent rules is deterministic, then the *Chain Loop* conflict is deterministic; otherwise, it is conditional.

Behavior Violation Detection. IoT systems often involve many devices with varying functionalities. By defining behavior properties, users can specify how these devices should not interact through rules and chains, ensuring the system functions as intended under different conditions. For instance, a fire door should not close when a smoke alarm is triggered. We define each behavior property as "Don't [ACTION] when [CONDITION]", including behavior properties from prior work [6], [9], [13], [20], as detailed in Table 12. For users without programming experience, we provide a user interface (Figure 6(c)) that simplifies defining these behavior properties. A behavior violation happens when executing the rules or chains yields results that contradict the defined properties. To detect violations, we build a rule interaction graph that captures all the rules. Then, for each behavior property, take its condition and trace possible paths in the interaction graph to see if the corresponding action can occur. If path exists, the system exhibits a behavior violation.

4.4. Conflict Resolver

Conflicts Penalty. Table 3 assigns rankings to conflicts based on their potential harm, providing a basis for prioritizing mitigation. If a conflict directly causes clear and irreversible harm to system security or physical safety, it is categorized as *high risk*. For instance, *Contradictory Action* may cancel out opposing actions and lead to unpredictable effects. *Accumulation Impact* increases security or safety risks by amplifying effects on shared physical channels, such as raising temperature, intensifying noise, or overloading electrical capacity.

Conflicts that have the potential to cause serious harm but do not invariably lead to the worst-case outcome are classified as *medium risk*. For example, *Duplicate Actions* may be highly dangerous in healthcare settings but may only result in resource waste in routine tasks. *Force Action* becomes severe only when it leads to critical operations. Similarly, *Chain Enable* can escalate risk by unintentionally triggering unsafe actions, while *Chain Loop* activates multiple rules in a cycle. Both require all rules in the chain to be triggered and typically do not cause immediate harm.

Conflicts that result in manageable issues, such as minor inconvenience, reduced efficiency, or limited functionality, are considered *low risk*. *Reduction Impact* diminishes physical effects rather than intensifying them, thus generally posing less danger. *Disorder Action* typically disrupts the intended sequence of actions without creating significant safety hazards. *Block Action* and *Chain Disable* can mitigate harm by preventing risky actions but may cause only minor inconvenience when blocking harmless actions.

To evaluate conflicts within their specific context, we introduce a *conflict penalty* metric to quantify the associated

TABLE 6: Accuracy comparison with IoTSan [21], Soteria [13], IoTCom [9], TAPInspector [6] and TAPFixer [20].

	IoTSan	Soteria	IoTCom	TAPInspector	TAPFixer	InteractionShield
#1	●	●	●	●	●	●
#2	○	◐	◐	◐	◐	●
#3	○	●	●	●	●	●
#4	○	○	◐	◐	◐	●
#5	○	○	◐	◐	◐	●
#6	○	○	◐	◐	◐	●
#7	○	○	○	◐	◐	●
#8	○	○	○	◐	◐	●

Note: ●: detected; ◐: partially detected; ○: not detected.

TABLE 7: Accuracy comparison with HomeGuard [8].

	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10
HomeGuard	◐	◐	◐	◐	◐	●	◐	◐	◐	◐
InteractionShield	●	●	●	●	●	●	●	●	●	●

Note: ●: detected; ◐: partially detected; ○: not detected.

risk. Suppose each action has a base risk score S, any action that violates one or more behavior properties within a chain increases its overall risk by the number of violations. In a scenario where an action appears in multiple chains and violates B behavior properties, its final risk score is $B + S$. When a conflict involves multiple actions, the combined action risk is determined by averaging the final scores of these actions. The conflict penalty P is computed by multiplying the conflict ranking R by this average score as $P = R \times \frac{1}{n} \sum_{i=1}^{n} \left(S_i + B_i\right)$. The total system penalty aggregates all individual conflict penalties. By combining both the action's risk and the conflict's severity, this metric provides a weighted measure of overall impact, supporting effective mitigation strategies.

Optimization-based Conflict Handling. In large-scale rule sets, completely eliminating conflicts is often infeasible and, in many cases, unnecessary. Conflicts of minor significance can be tolerated, particularly when their associated risks are relatively low. We use optimization strategies to contain risks and strike an acceptable compromise. Our main goal is to identify a configuration of rules that preserves an acceptable risk level while maximizing system functionality, keeping as many useful, valid rules as possible. This approach yields a realistic system design, where certain conflicts may remain but do not severely compromise critical properties, and their impact can be minimized through optimization techniques.

To resolve conflicts under these constraints, we define an objective function that seeks to maximize the number of rules while minimizing the overall conflict penalty score. Formally, this objective function is:

$$\text{Minimize} \sum_{j \in \text{violations}} \left(P_j \times t_j\right) - \lambda \times \sum_{i \in \text{rules}} c_i$$

Where P_j is the conflict penalty, t_j indicates whether the conflict is deterministic ($t_j = 1$) or conditional($t_j = 0.5$), c_i is a binary variable indicating whether rule i is included in the final system, and λ is a tuning parameter that adjusts the trade-off between minimizing conflict penalties and maximizing the number of active rules. We use a genetic algorithm to solve this optimization problem.

5. Evaluations

We implemented a prototype of InteractionShield and evaluated its performance through conflict detection and resolution experiments. All experiments were conducted on a machine with an Intel i7-12700K processor and 64 GB of RAM, under a medium-risk-level configuration. For the case study, we assigned devices according to their descriptions, while in the other experiments, we identified all possible device connections and randomly selected one to simulate typical real-world deployments.

5.1. TestBed Setting

Test Apps. To compare InteractionShield with existing approaches, we used three distinct datasets. The first is the SmartHomeBench dataset from TAPInspector [6], which consists of eight groups of smart home interactions: Groups 1-3 from Soteria [13], Groups 4-6 from IoTCom [9], and Groups 7-8 created by TAPInspector [6] and TAPFixer [20]. The second consists of app groups from HomeGuard [8]. IoTCom and IoTSan support only a limited number of real-world apps. To ensure a fair comparison, we generated 100 SmartApps per experiment, each containing a single rule that could be successfully analyzed by all three tools.

Real-World Apps. To assess performance under realistic conditions, we used 2,101 SmartApps from the SmartAppZoo dataset [22], 2,788 IFTTT applets from [23], and 2,086 OpenHAB rules sourced from third-party apps on GitHub, resulting in a total of 6,975 apps. Real-world IoT apps typically involve many devices and complex interactions, so this dataset tests InteractionShield's ability to detect and resolve conflicts at scale.

5.2. Accuracy Validation

We validated the accuracy of InteractionShield by conducting a direct comparison with two publicly available tools: IoTCom [9] and IoTSan [21]. For a broader comparison with other existing works, we relied on their reported results. The overall detection accuracy results are summarized in Tables 6 and 7. Tables 16 and 17 present detailed app descriptions and the complete results identified by our tool.

As an example, Table 6 presents two apps in Group 7. App 1 includes two rules: one to turn off the heater when the temperature exceeds 30°C, and another to turn it on when the temperature falls below 20°C. This setup creates a *Chain Loop* conflict: once the heater turns on, causing the temperature to rise and eventually surpass 30°C, which turns the heater off; when the temperature subsequently falls below 20°C again, the heater is turned back on. App 2 in this group switches off a smart plug, yet the heater should be turned off before the plug to avoid a *Disorder Action* conflict. In Group 8, rules that turn on the heater while opening a window lead to a *Reduction Impact* conflict. Additionally, a similar *Chain Loop* is created when rules to turn the heater on or off and open or close a window affect the room temperature. All of these conflicts were missed by previous works.

Turning to Table 7, HomeGuard overlooks every *Duplication Action* conflict, such as those found in Groups 2 and 7. Since the apps in Groups 3, 5, 7, 8, 9 and 10 are labeled as *Convenience*, their actions carry a risk score lower than medium. The chains in these groups do not actually trigger *Chain Enable* or *Chain Disable* conflicts, yet HomeGuard still flags them as such. Furthermore, HomeGuard fails to detect several other critical conflicts. For example, in Group 1, turning on a fan can cause the temperature to fall below 72°F, triggering a *Chain Enable* conflict. In Group 3, running both a heater and an oven rapidly increases the ambient temperature, producing an *Accumulation Impact* conflict. In Group 4, turning on a fan can cause the overall power consumption to exceed a 3000W threshold. As the fan is then turned off, the resulting temperature increase may trigger the fan to turn on again, creating a *Chain Loop* conflict. In Group 5, turning on a humidifier and turning off a vent increases humidity, also forming an *Accumulation Impact*. In Group 7, rules that activate lights to turn on and off simultaneously amount to both a *Duplication Action* and *Contradictory Action* conflict. HomeGuard fails to detect any of these conflicts.

5.3. Conflict Detection

To compare InteractionShield with IoTCom and IoTSan, we generated random SmartApps, grouped them in sets ranging from 5 to 25, and configured each group according to the requirements of each tool.

Figure 2 presents the results of the detection comparison: Figure 2(a) shows the average detection time across different app sizes. Leveraging an efficient graph search algorithm, InteractionShield avoids the time-consuming model checks required by other approaches. As a result, it analyzes 25 single-rule apps in only 0.05 seconds, compared to 242.98 seconds for IoTCom and 278.98 seconds for IoTSan, achieving a reduction of over 99% in detection time.

Figure 2(b) further shows that our tool identifies the largest number of potential conflicts, demonstrating promising detection accuracy and coverage. While IoTCom occasionally detects more *Chain Enable* and *Chain Disable* conflicts for smaller app groups, this is because it considers all possible rule connections as conflicts, without distinguishing whether the interactions lead to harmful outcomes. In contrast, InteractionShield classifies a connection as a conflict only if the action has a risk level greater than *medium*. Moreover, IoTCom labels actions with the same capability as *Duplication*, but this approach can be misleading: the same capability may connect to different devices, and thus some duplications reported by IoTCom are not genuine. Overall, InteractionShield detects the most numerous and diverse types of conflicts, highlighting its efficiency and effectiveness.

To evaluate the large-scale conflict detection capabilities of InteractionShield, we conducted experiments using a dataset of 6,975 real-world IoT apps. In each experiment, we randomly selected groups of 5 to 25 apps from the dataset and assumed that all physical connections were valid. Each experiment was repeated 2,000 times to calculate the average

Figure 2: Comparison of different tools. (a) Average time consumption; (b) Total number of conflicts detected.

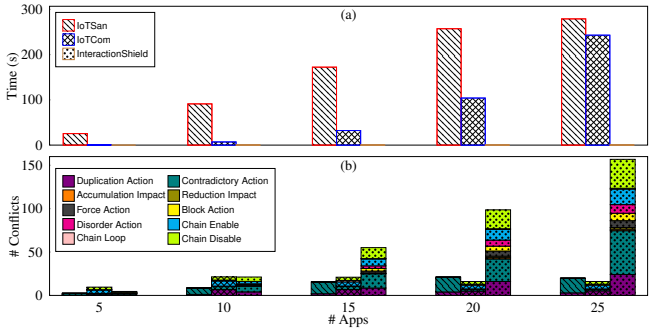
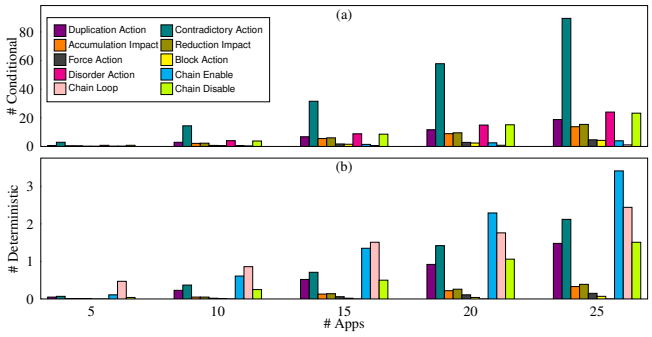


Figure 3: The average number of conflicts per group. (a) Conditional conflicts; (b) Deterministic conflicts.



number of conflicts per group. The results, summarized in Figure 3, provide an overview of the average number of detected conflicts across all 10 conflict types. Specifically, Figure 3(a) highlights *conditional conflicts*, while Figure 3(b) focuses on *deterministic conflicts*. Our findings show that *Contradictory Action* is the most frequently detected interaction threat, with deterministic rule conflicts occurring far less often than conditional ones. Among deterministic conflicts, *Chain*-related conflicts are the most prevalent, which aligns with our expectation given the assumption that all physical connections are valid, making deterministic interactions more common. We also discovered conflicts caused by *Force Action*, a type of conflict not previously reported in other studies. Our analysis revealed that approximately 2.27% of all conflicts can be attributed to *Force Action*. Finally, to validate our results, we randomly selected a 20 app group containing a total of 87 identified rule conflicts, manually reviewed and confirmed the accuracy of every detected conflict.

5.4. Conflict Handling

To resolve conflicts effectively, it is essential to determine the optimal value of the parameter λ in the conflict-handling objective function. λ controls the trade-off between preserving rules and avoiding conflicts. If λ is too small, the system may remove excessive rules, reducing functionality; if too large, it may tolerate too many conflicts, leading to unpredictable or unsafe behavior. Finding the right balance ensures an optimal trade-off between stability and flexibility.

Figure 4: Estimation of λ from rule count and penalty change.

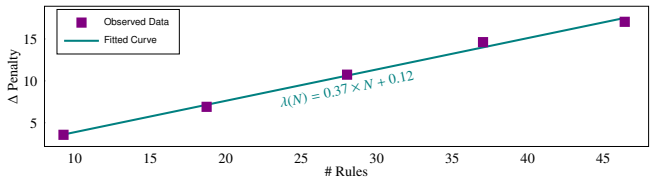
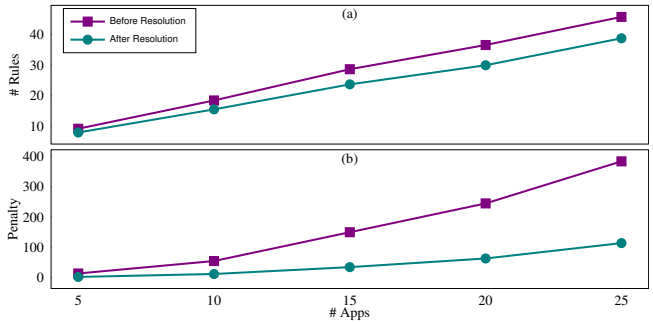


Figure 5: Impact of conflict resolution on (a) Rule count and (b) Penalties.



We describe this balance in rule addition using the concepts of marginal cost and marginal benefit. The marginal cost is the additional penalty incurred when a new rule is added, denoted as $\Delta Penalty$, which captures the extra conflicts introduced with existing rules. The marginal benefit is the extra reward gained from adding a new rule, denoted as λ. If $\Delta Penalty > \lambda$, the optimizer favors removing the rule; if $\Delta Penalty < \lambda$, it favors keeping the rule. Equilibrium is reached when $\lambda \approx \Delta Penalty$, meaning the net effect of adding a rule is zero. At this point, λ captures the precise trade-off, allowing the system to balance competing objectives and achieve scalable conflict resolution without overfitting or underfitting rules.

To estimate the value of λ, we calculate the penalty change incurred by adding a single rule to each group. Specifically, we analyze five groups of randomly selected apps, each containing between 5 and 25 apps with varying numbers of rules. For each group, we first measure the overall penalty, then add one rule, re-measure the penalty, and compute the penalty change. We record the average number of rules per group against the corresponding penalty change. This process is repeated 500 times to ensure robust results. Finally, we plot the average number of rules against the average penalty change in Figure 4. The results show that the penalty change is linearly related to the number of rules, based on which, we further derived that $\lambda(N) = 0.37 \times N + 0.12$, where N is the number of rules.

Figure 5 illustrates the effect of conflict resolution on rule count and penalties. As the number of rules increases, the total penalty also rises. After applying our conflict resolution algorithm, penalties drop significantly while the number of valid rules remains close to the maximum, with only a slight reduction. These results demonstrate that the proposed approach effectively mitigates conflicts while preserving functionality, ensuring a stable and efficient smart home environment.

TABLE 8: Details of example apps in Group 4.

Rule	Category	Trigger	Action
r_1	Safety&Security	presence.not present	location.Home
r_2	Safety&Security	location.Home	oven.heating
r_3	Safety&Security	smoke.detected	door.open

5.5. Case Study

To illustrate the conflict-handling process, we present case studies using three apps from Group 4 in Table 6, with details provided in Table 8. A randomly generated layout of the devices associated with these apps is shown in Figure 6(b). Risk scores and conflict rankings are represented numerically as follows: 3 for *high*, 2 for *medium*, and 1 for *low*. The initial environment is configured with a room size of $5m \times 4m \times 3m$, an ambient temperature of 70°F, an oven temperature of 450°F, a smoke generation rate of 50 mg/min, and an initial smoke concentration of 0 mg/m^3. In addition, we specify a predefined behavior property: *DON'T turn on the oven WHEN no one is at home*.

The detection results are summarized in Table 9 and Figure 6(d). All three apps belong to the *Safety & Security* category. Accordingly, the action *oven.heating* is assigned a high risk score, while *door.open* is assigned a low risk score. These two actions have opposite physical effects through both the smoke and temperature channels. Moreover, action *oven.heating* generates smoke, which influences the trigger *smoke.detected*, creating a link between r_2 and r_3. Since these are physical interactions, we apply the flow functions of the temperature and smoke channels to confirm their validity. This interaction chain also violates the predefined behavior property, resulting in one violation attributed to *oven.heating*. The interaction between *oven.heating* and *door.open* forms a *Reduction Impact* conflict, denoted as L_1. Its conflict ranking is low, and the overall risk score is the average of the risk scores and violations of the two actions involved, resulting in a conflict penalty of 2.5 for L_1.

For conflict L_2, r_1 and r_2 are linked through the cyber channel *location.Home*. Since *oven.heating* is a high-risk action, the chain constitutes a *Chain Enable* conflict. As this action also violates the predefined behavior property, its overall risk score increases to 4. With a medium conflict ranking, the penalty for L_2 is 8.

For conflict L_3, rule r_2 is linked with rule r_3 through the smoke channel. However, because *door.open* has a low risk score, this is not considered a genuine conflict.

To resolve these interaction threats, we remove the conflicting rules. Since both genuine conflicts are conditional, the weighted penalty score for the current set of rules is 5.25. With $\lambda = 1.23$, the value of the objective function is 1.56. After removing rule r_2, the penalty score drops to 0 while preserving the maximum number of valid rules, yielding an optimal objective function value of -2.46.

TABLE 9: Detection results of interaction threats for example apps.

No.	Type	Ranking	Rules	Action	Risk	Violation	Penalty
L_1	**C.4**	Low	r_2 r_3	over.heating door.open	High Low	1 0	2.5
L_2	**C.8**	Medium	$r_1 \rightarrow r_2$	oven.heating	High	1	8
L_3	**C.8**	Medium	$r_2 \rightarrow r_3$	door.open	Low	0	2

TABLE 10: Performance evaluation of system components.

# App	Avg # Rules	Time(s)/Memory(MB)		
		Conflicts Detector	Conflicts Resolver	Total
5	9	0.21/0.11	14.69/0.31	14.90/0.42
	10	0.26/0.12	19.68/0.31	19.94/0.43
25	45	8.66/1.50	714.32/1.48	722.98/2.98
	46	8.90/1.52	742.81/1.53	751.70/3.05

5.6. Performance Overhead

To evaluate the performance overhead of InteractionShield, we measured the average time and memory consumption of the Conflict Detector and Resolver across randomly selected groups containing 5 to 25 apps. As shown in Table 10, InteractionShield required an average of 14.9 seconds for groups with 5 apps and 723 seconds for groups with 25 apps (averaged over 500 runs) to detect and resolve all rule conflicts. Notably, the Detector was significantly faster than the Resolver. We further measured the overhead of adding a new rule. In both settings, a single added rule resulted in only a slight increase in time and memory usage. These results demonstrate that the system scales efficiently, maintaining low overhead even as rule complexity increases.

6. Related work

IoT security has been extensively studied from multiple aspects, including firmware security [1], [2], traffic analysis [3], [4], physical vulnerabilities [19], [24], [25], overprivilege issues [26], device vulnerabilities [27], and policy enforcement [28]–[30].

In particular, significant attention has been given to rule interactions [5]–[15], [31]–[33]. For instance, [5] focused on detecting cross manual-control interaction threats in multi-platform, multi-control-channel home environments, [6], [7] worked on detecting interaction vulnerabilities in delay-based automation scenarios, [8]–[10], [13] explored rule interference through static code analysis, [11], [31] conducted conflict analysis in IFTTT based on text analysis, [10] proposed methods for enforcing policies on rule conflicts in real-time scenarios. Additionally, several studies [12], [14], [15], [31]–[33] have defined various types of rule conflicts, contributing to the overall understanding of rule interaction vulnerabilities.

7. Limitations and Discussion

Properties Conflicts. Resolving rule conflicts often requires balancing competing priorities, such as comfort, energy efficiency, and security. Users play a crucial role in determining the right balance based on their individual needs. To accommodate this, we allow users to define their own behavior properties, assuming they will provide accurate and meaningful inputs. However, inexperienced users may struggle to express their preferences correctly or may inadvertently introduce conflicting properties. Currently, we do not address conflicts within user-provided behavior properties. In future work, we plan to incorporate mechanisms to detect and resolve such conflicts.

Device Context. Our work focuses on widely used devices across major IoT platforms, but newly introduced device categories may not yet be covered. Moreover, device risk is highly context-dependent. For example, a light in the bedroom may pose a different level of risk than one in the living room. Our current approach does not capture this level of granularity.

8. Conclusion

In this paper, we present InteractionShield, a framework for detecting and resolving interaction threats in smart homes. InteractionShield analyzes event relations and categorizes rule interferences into different types. By accurately modeling these relations, InteractionShield identifies rule conflicts, enabling accurate identification of rule conflicts. Conflicts are further assessed through risk scoring and ranking, and resolved using an optimization-based approach. To validate its effectiveness, we implement a prototype of InteractionShield and evaluate it on a large-scale dataset of real-world IoT apps. Experimental results demonstrate that InteractionShield accurately detects and resolves interaction threats while incurring minimal overhead.

Acknowledgment

Zhaohui Wang, Bo Luo, and Fengjun Li were supported in part by NSF IIS-2014552, DGE-1565570, and the Ripple University Blockchain Research Initiative. The authors would like to thank the anonymous reviewers and the shepherd for their valuable comments and suggestions.

References

[1] P. Sun, L. Garcia, G. Salles-Loustau, and S. Zonouz, "Hybrid firmware analysis for known mobile and IoT security vulnerabilities," in *2020 50th annual IEEE/IFIP international conference on dependable systems and networks (DSN)*, pp. 373–384, IEEE, 2020.

[2] B. Zhao, S. Ji, J. Xu, Y. Tian, Q. Wei, Q. Wang, C. Lyu, X. Zhang, C. Lin, J. Wu, *et al.*, "A large-scale empirical analysis of the vulnerabilities introduced by third-party components in IoT firmware," in *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*, pp. 442–454, 2022.

[3] A. Acar, H. Fereidooni, T. Abera, A. K. Sikder, M. Miettinen, H. Aksu, M. Conti, A.-R. Sadeghi, and S. Uluagac, "Peek-a-boo: I see your smart home activities, even encrypted!," in *ACM WiSec*, 2020.

[4] Y. Luo, L. Cheng, H. Hu, G. Peng, and D. Yao, "Context-rich privacy leakage analysis through inferring apps in smart home IoT," *IEEE Internet of Things Journal*, pp. 2736–2750, 2020.

[5] H. Chi, Q. Zeng, and X. Du, "Detecting and handling IoT interaction threats in multi-platform multi-control-channel smart homes," in *32nd USENIX Security Symposium*, pp. 1559–1576, 2023.

[6] Y. Yu and J. Liu, "TAPInspector: Safety and liveness verification of concurrent trigger-action IoT systems," *IEEE Transactions on Information Forensics and Security*, vol. 17, pp. 3773–3788, 2022.

[7] H. Chi, C. Fu, Q. Zeng, and X. Du, "Delay wreaks havoc on your smart home: delay-based automation interference attacks," in *2022 IEEE Symposium on Security and Privacy*, pp. 285–302, IEEE, 2022.

[8] H. Chi, Q. Zeng, X. Du, and J. 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)*, pp. 411–423, IEEE, 2020.

[9] M. Alhanahnah, C. Stevens, and H. Bagheri, "Scalable analysis of interaction threats in IoT systems," in *Proceedings of the 29th ACM SIGSOFT international symposium on software testing and analysis*, pp. 272–285, 2020.

[10] Z. B. Celik, G. Tan, and P. D. McDaniel, "IoTGuard: Dynamic enforcement of security and safety policy in commodity IoT.," in *NDSS*, 2019.

[11] Q. Wang, P. Datta, W. Yang, S. Liu, A. Bates, and C. A. Gunter, "Charting the attack surface of trigger-action IoT platforms," in *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*, pp. 1439–1453, 2019.

[12] W. Brackenbury, A. Deora, J. Ritchey, J. Vallee, W. He, G. Wang, M. L. Littman, and B. Ur, "How users interpret bugs in trigger-action programming," in *Proceedings of the 2019 CHI conference on human factors in computing systems*, pp. 1–12, 2019.

[13] Z. B. Celik, P. McDaniel, and G. Tan, "Soteria: Automated IoT safety and security analysis," in *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pp. 147–158, 2018.

[14] F. Corno, L. De Russis, and A. Monge Roffarello, "Empowering end users in debugging trigger-action rules," in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pp. 1–13, 2019.

[15] M. Ma, S. M. Preum, and J. A. Stankovic, "CityGuard: A watchdog for safety-aware conflict detection in smart cities," in *Proceedings of the Second International Conference on Internet-of-Things Design and Implementation*, pp. 259–270, 2017.

[16] M. Chen, Y. Ma, K. Song, Y. Cao, Y. Zhang, and D. Li, "Improving large language models in event relation logical prediction," in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9451–9478, 2024.

[17] J. F. Allen, "Maintaining knowledge about temporal intervals," *Communications of the ACM*, vol. 26, no. 11, pp. 832–843, 1983.

[18] Z. Wang, B. Luo, and F. Li, "PrivacyGuard: Exploring hidden cross-app privacy leakage threats in IoT apps," in *Proceedings on Privacy Enhancing Technologies*, pp. 776–791, 2025.

[19] M. O. Ozmen, X. Li, A. Chu, Z. B. Celik, B. Hoxha, and X. Zhang, "Discovering IoT physical channel vulnerabilities," in *ACM Conference on Computer and Communications Security*, pp. 2415–2428, 2022.

[20] Y. Yu, Y. Xu, K. Huang, and J. Liu, "TAPFixer: Automatic detection and repair of home automation vulnerabilities based on negated-property reasoning," in *33rd USENIX Security Symposium*, pp. 4945–4962, USENIX Association, 2024.

[21] D. T. Nguyen, C. Song, Z. Qian, S. V. Krishnamurthy, E. J. Colbert, and P. McDaniel, "IoTSan: Fortifying the safety of IoT systems," in *Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies*, pp. 191–203, 2018.

[22] Z. Wang, B. Luo, and F. Li, "SmartAppZoo: a repository of smartthings apps for IoT benchmarking," in *the 8th ACM/IEEE Conference on Internet of Things Design and Implementation*, pp. 448–449, 2023.

[23] H. Yu, J. Hua, and C. Julien, "Analysis of IFTTT recipes to study how humans use internet-of-things IoT devices," in *19th ACM Conference on Embedded Networked Sensor Systems*, pp. 537–541, 2021.

[24] W. Ding and H. Hu, "On the safety of IoT device physical interaction control," in *ACM CCS*, pp. 832–846, 2018.

[25] W. Ding, H. Hu, and L. Cheng, "IOTSAFE: Enforcing safety and security policy with real IoT physical interaction discovery," in *NDSS*, 2021.

[26] E. Fernandes, J. Jung, and A. Prakash, "Security analysis of emerging smart home applications," in *IEEE S&P*, pp. 636–654, IEEE, 2016.

[27] L. Babun, H. Aksu, and A. S. Uluagac, "A system-level behavioral detection framework for compromised cps devices: Smart-grid case," *ACM TCPS*, pp. 1–28, 2019.

[28] 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 *USENIX Security*, 2017.

[29] T. Gu, Z. Fang, A. Abhishek, H. Fu, P. Hu, and P. Mohapatra, "IoTGaze: IoT security enforcement via wireless context analysis," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, pp. 884–893, IEEE, 2020.

[30] M. Yahyazadeh, S. R. Hussain, E. Hoque, and O. Chowdhury, "Patriot: Policy assisted resilient programmable IoT system," in *Runtime Verification: 20th International Conference, Los Angeles, CA, USA, October 6–9, 2020, Proceedings 20*, pp. 151–171, Springer, 2020.

[31] B. Huang, H. Dong, and A. Bouguettaya, "Conflict detection in IoT-based smart homes," in *2021 IEEE International Conference on Web Services (ICWS)*, pp. 303–313, IEEE, 2021.

[32] Y. Fu, D. Zhang, C. Wang, X. Luo, W. Liu, T. Luo, and F. Fang, "Conflict detection of device linkage rules in smart home systems," in *2020 Chinese Automation Congress*, pp. 3160–3166, IEEE, 2020.

[33] C.-J. M. Liang, L. Bu, Z. Li, J. Zhang, S. Han, B. F. Karlsson, D. Zhang, and F. Zhao, "Systematically debugging IoT control system correctness for building automation," in *Proceedings of the 3rd ACM international conference on systems for energy-efficient built environments*, pp. 133–142, 2016.

[34] H. Ibrhim, S. Khattab, K. Elsayed, A. Badr, and E. Nabil, "A formal methods-based rule verification framework for end-user programming in campus building automation systems," *Building and Environment*, vol. 181, p. 106983, 2020.

[35] H. Lin, C. Li, J. Lang, Z. Wang, L. Fan, and C. Duan, "CP-IoT: A cross-platform monitoring system for smart home," in *Proc. Netw. Distrib. Syst. Secur.(NDSS) Symp*, pp. 1–18, 2024.

[36] Google, "Smart Home Device Types." https://developers.home.google.com/cloud-to-cloud/guides, 2025. [Online; accessed 10-May-2025].

[37] Apple, "Home accessories." https://www.apple.com/home-app/accessories/, 2025. [Online; accessed 10-May-2025].

[38] SmartThings, "Production Capabilities." https://developer.smartthings.com/docs/devices/capabilities/capabilities-reference, 2025. [Online; accessed 10-May-2025].

[39] Y. Sun, X. Wang, H. Luo, and X. Li, "Conflict detection scheme based on formal rule model for smart building systems," *IEEE Transactions on Human-Machine Systems*, vol. 45, no. 2, pp. 215–227, 2014.

[40] A. Al Farooq, E. Al-Shaer, T. Moyer, and K. Kant, "IoTC^2: A formal method approach for detecting conflicts in large scale IoT systems," in *2019 IFIP/IEEE symposium on integrated network and service management (IM)*, pp. 442–447, IEEE, 2019.

[41] D. N. Mekuria, P. Sernani, N. Falcionelli, and A. F. Dragoni, "Consistency verification of a rule-based smart home reasoning system with satisfiability modulo theories," in *2020 16th International Conference on Intelligent Environments (IE)*, pp. 52–59, IEEE, 2020.

[42] Y. Xing, L. Hu, X. Du, Z. Shen, J. Hu, and F. Wang, "CCDF-TAP: A context-aware conflict detection framework for IoT trigger-action programming with graph neural network," *IEEE Internet of Things Journal*, 2024.

[43] D. Han, G. Ma, X. Zhang, Y. Cai, and A. Li, "Rule conflict classification and detection for smart building systems: A case study," in *43rd Chinese Control Conference*, pp. 5153–5158, IEEE, 2024.

[44] A. Kashaf, V. Sekar, and Y. Agarwal, "Protecting smart homes from unintended application actions," in *ACM/IEEE International Conference on Cyber-Physical Systems*, pp. 270–281, IEEE, 2022.

Appendix A.
Existing Work on Conflict Detection

A.1. Conflict Definitions

We map conflict definitions from existing studies into our framework, covering nine out of the ten conflict types, only the *Force Action* conflict does not appear in prior work. A key challenge is the lack of consistent terminology across the literature, where the same conflict type is often described under different names or definitions. Our framework resolves this by providing a unified mapping, which not only organizes existing research but also highlights commonalities and differences.

TABLE 11: Mapping existing work to our definition.

Name	Previous Work
Duplication Action	Action-Action Repeat [9], Exclusive Event Coordination [9], Condition Inconsistency Conflict [34], Opposite Execution Actions Conflict [32], Conditional Mutex Conflict [32], Same Repeated Attributes [10], [13], Action Duplicate [11], [35], Redundancies [14], Inconsistent Events [13], Rule Dependency Conflict [34].
Contradictory Action	Action-Action Conflict [9], Inconsistencies [14], Numeric Device Conflict [15], Opposite Device Conflict [15], Race Condition [5], Potential Race Condition [5], Action Revert [5], [11], [35], Action Conflict [7], [11], [35], Function-Function Conflict [31], Action Interference Threats [8], Value Inconsistency Conflict [34], Non-Specified Conflict [34], Attributes of Conflicting Values [10], [13], Race Condition of Events [10], [13], Contradictory Action [12], Conflict Device Commands [33].
Accumulation Impact	Cumulative Environment Impact Conflict [31], Additive Environmental Conflict [15].
Reduction Impact	Opposite Environment Impact Conflict [31], Value Inconsistency Conflict [34], Non-Specified Conflict [34], Environmental Conflict [32], Contradictory Action [12], Opposite Envionmental Conflict [15].
Block Action	Action Breaking [6], Device disabling [6].
Disorder Action	Disordered Action Scheduling [6], Action Overriding [6], Condition Dynamic Blocking [6], Scheduled Condition Blocking [6], Action Disordering Attack [7], Condition Disabling Attack [7], Condition Enabling Attack [7], Delayed Parallel Execution [7], Delayed Chained Execution [7], Nondeterministic Timing [12], Duration Device Conflict [15], Dependent Environmental Conflict [15].
Chain Enable	Condition Enabling [5], Chained Execution [5], Chained Execution [7], Overlapping Attack [7], Transitive Envionment Impact Conflict [31], Action Trigger [9], Action-Condition Match [9], Trigger Interference Threats [8], Condition Interference Threats [8].
Chain Loop	Infinite Loop [5], [7], [12], Self Coordination [9], Rule Loop Conflict [32], Cycle of Device Attributes [10], Action Loop [11], [35], Loops [14].
Chain Disable	Condition Disabling [5], Tardy-channel-based Rule Blocking [6], Condition Dynamic Blocking [6], Scheduled Condition Blocking [6], Device disabling [6], Action-Condition No Match [9], Condition Interference Threats [8], Condition Block [11].

A.2. Behavior Properties

Table 12 provides examples of behavior properties drawn from existing studies, illustrating how behavior property of the form "Don't [ACTION] when [CONDITION]" are expressed in practice.

TABLE 12: Examples of behavior properties.

ID	Behavior Property Description
P.1	DON'T unlock the door WHEN the user is not home.
P.2	DON'T turn on the oven WHEN the user is not home.
P.3	DON'T disarm the security camera WHEN the user is not home.
P.4	DON'T play music WHEN the location mode is sleeping.
P.5	DON'T turn on the light WHEN the location mode is sleeping.
P.6	DON'T turn on the air conditioner WHEN the location mode is away.
P.7	DON'T turn on the water valve WHEN a leak is detected.
P.8	DON'T unlock the door WHEN smoke is detected.
P.9	DON'T turn on the heater WHEN the window is open.
P.10	DON'T turn on the light WHEN the time is later than 11 p.m.

Appendix B.
Detailed Design of InteractionShield

B.1. Formal Computation of Event Interference

For each relation r drawn from the set $\{L, M, S, F\}$, define the disjunction $r_i \vee r_j$ as the set $\{r_i, r_j\}$, and define the conjunction $r_i \wedge r_j$ as follows: if $r_i \neq r_j$, then $r_i \wedge r_j = \varnothing$; if $r_i = r_j$, then $r_i \wedge r_j = r_i$ (which is equal to r_j). Concretely, when two relations differ, their disjunction simply collects both relations, while their conjunction becomes empty. However, when the two relations are the same, their disjunction remains the single relation, and their conjunction likewise remains that same relation.

Given an event relation $\rho_i = \langle L_i, M_i, S_i, F_i \rangle$, the disjunction and conjunction are defined coordinate-wise as follows:

$$\bigvee_{i=1}^{I} \rho_i = \langle \bigvee_{i=1}^{I} L_i, \bigvee_{i=1}^{I} M_i, \bigvee_{i=1}^{I} S_i, \bigvee_{i=1}^{I} F_i \rangle$$

$$\bigwedge_{i=1}^{I} \rho_i = \langle \bigwedge_{i=1}^{I} L_i, \bigwedge_{i=1}^{I} M_i, \bigwedge_{i=1}^{I} S_i, \bigwedge_{i=1}^{I} F_i \rangle$$

The trigger interference between two triggers can be calculated as follows:

$$\begin{aligned}\tau(E(T_i), E(T_j)) &= \tau\left(\bigvee_{i=1}^{I}\left(\bigwedge_{m=1}^{M_i} e_{i,m}\right), \bigvee_{j=1}^{J}\left(\bigwedge_{n=1}^{N_j} e_{j,n}\right)\right) \\ &= \bigvee_{i=1}^{I}\left(\bigwedge_{m=1}^{M_i}\left[\bigvee_{j=1}^{J}\left(\bigwedge_{n=1}^{N_j} \rho(e_{i,m}, e_{j,n})\right)\right]\right)\end{aligned}$$

The action interference between two actions can be calculated as:

$$\tau(E(A_i), E(A_j)) = \tau(e_i, e_j) = \rho(e_i, e_j)$$

The chain interference between an action and a trigger is calculated as:

$$\tau(E(A_i), E(T_j)) = \tau\left(e_i, \bigvee_{j=1}^{J}\left(\bigwedge_{n=1}^{N_j} e_{j,n}\right)\right) = \bigvee_{j=1}^{J}\left(\bigwedge_{n=1}^{N_j} \rho\left(e_i, e_{j,n}\right)\right)$$

B.2. Device Clusters

We first collected a list of commonly used devices from popular IoT platforms, including Google Smart Home, Apple HomeKit, and SmartThings [36]–[38], and grouped them into clusters following the device clusters in [18]. Since *Appliances* is a broad category, we further separated devices with notable safety risks into more specific clusters. For example, ovens and stoves were assigned to the *Fire* category, while faucets and valves were placed in the *Water* category.

TABLE 13: Clusters of commonly used devices.

Cluster	Devices
Activity	activity, bathtub, bed, shower, sleep
Alarm	alarm
Appliance	appliance, blender, blind, cleaner, A/C, cooler, curtain, dishwasher, dryer, fan, freezer, light, mop, mower, printer, projector, refrigerator, washer, TV, thermostat, vacuum
Battery	battery
Car	car, vehicle
Door	door, garage, gate, window
Fire	coffee, cooker, cooktop, fireplace, fryer, heater, kettle, microwave, oven, stove
Fitness	step, watch, wristband
Gas	gas
Health	body, health, medicine
Location	geolocation
Lock	lock
Monitoring	camera, audio, image, speech, video
Music	player, soundbar, speaker
Power	outlet, power
Presence	location mode, occupancy, presence
Smoke	dioxide, monoxide, smoke
Water	faucet, valve, water, sprayer, sprinkle

B.3. Risk Score for Device Clusters

Safety risk refers to the potential for physical harm, property damage, or critical system failures. Privacy risk concerns the unauthorized collection, use, or disclosure of personal data. Convenience risk relates to service disruptions, loss of functionality, or added effort caused by device malfunctions. Each risk is classified as high, medium, or low. High risk denotes a significant likelihood of severe impact, often threatening life or property. Medium risk indicates a moderate impact, less severe or less likely, but still notable. Low risk suggests minimal consequences that are rare, minor, and generally manageable.

Safety Risks. For smart devices, safety risk is often either severe or negligible, placing most clusters into high or low categories. High risk arises when a device's state directly compromises protection or creates hazards. For example, active *Fire* or *Gas* devices can cause fire or explosion if faulty, a *Water* device failure may lead to flooding, and a locked *Lock* can block evacuation. Similarly, disabling safety-critical devices such as *Alarms*, *Monitoring* cameras, or *Smoke* detectors prevents timely alerts, leaving occupants vulnerable. Even a closed Door can obstruct emergency exits. By contrast, devices not tied to immediate hazards pose only low safety risk, regardless of state.

Privacy risks. Privacy risk varies with data sensitivity. High risk comes from devices that expose highly sensitive, identifiable, or security-critical information. Examples include *Car* and *Location* devices that reveal whereabouts, *Door* devices that indicate occupancy, *Fitness* and *Health* trackers that disclose medical data, and *Monitoring* devices that capture audio or video. Some risks persist even when devices are inactive. For instance, an unlocked *Lock* or a *Presence* sensor indicating absence can signal vulnerability. Medium risk applies to devices that reveal personal habits or preferences, such as *Activity* sensors, *Appliance* and *Fire* devices showing usage cycles, or *Music* systems reflecting routines. While such data can be exploited when combined with other information, it is less sensitive in isolation. Low risk is assigned to devices that collect only operational or environmental data with little connection to personal identity, such as *Smoke*, *Gas*, *Water*, or *Power* sensors, especially when inactive.

Convenience risks. Convenience risks reflect the impact of device failures on daily life such as disruptions of daily routines or essential services. High risk involves essential utilities whose loss severely disrupts living, for example, failure of *Gas* systems (cooking, heating), *Power* outlets (electricity supply), or *Water* services. Medium risk covers noticeable but manageable disruptions, such as non-functioning *Appliance* or *Fire* devices that reduce comfort, or dead *Batteries* that disable devices without halting daily life. Low risk applies to failures that cause only minor annoyance, such as a broken *Music* system that affects entertainment but not essential functions.

TABLE 14: Risk score for device clusters.

Cluster	Safety		Privacy		Convenience	
	On	Off	On	Off	On	Off
Activity	Low	Low	Medium	Low	Low	Low
Alarm	Low	High	Low	Low	Low	Low
Appliance	Low	Low	Medium	Low	Low	Medium
Battery	Low	Low	Low	Low	Low	Medium
Car	Low	Low	High	Low	Low	Low
Door	Low	High	High	Low	Low	Low
Fire	High	Low	Medium	Low	Low	Medium
Fitness	Low	Low	High	Low	Low	Low
Gas	High	Low	Low	Low	Low	High
Health	Low	Low	High	Low	Low	Low
Location	Low	Low	High	Low	Low	Low
Lock	High	Low	Low	High	Low	Low
Monitoring	Low	High	High	Low	Low	Low
Music	Low	Low	Medium	Low	Low	Low
Power	Low	Low	Low	Low	Low	High
Presence	Low	Low	Low	High	Low	Low
Smoke	Low	High	Low	Low	Low	Low
Water	High	Low	Low	Low	Low	High

B.4. Flow Functions

We build on the flow functions of different channels described in [19] and extend them by introducing a flow function for the *Power* channel. We also adopt the device sensitivity values and detection thresholds reported in [19]. The power flow function aggregates the instantaneous power

consumption of all devices:

$$P(t) = \sum_{i=1}^{N} P_i(t)$$

where $P_i(t)$ is the instantaneous power consumption of device i, and N is the total number of devices. A power conflict is detected when the aggregated power exceeds a predefined threshold P_{th}: $P(t) > P_{\text{th}}$.

B.5. InteractionShield GUI

Figure 6 illustrates the user-friendly graphical interface of InteractionShield, designed to support intuitive interaction and analysis. Panel (a), the app selector, allows users to choose specific IoT apps for evaluation. Panel (b) supports configuration of the room layout, including device positions, device types, and relevant environmental parameters. Panel (c) provides the interface for defining behavioral properties to prevent unwanted actions. Finally, panel (d) displays the threat detection results, offering a clear summary of identified risks and potential vulnerabilities.

Appendix C. Detection Performance

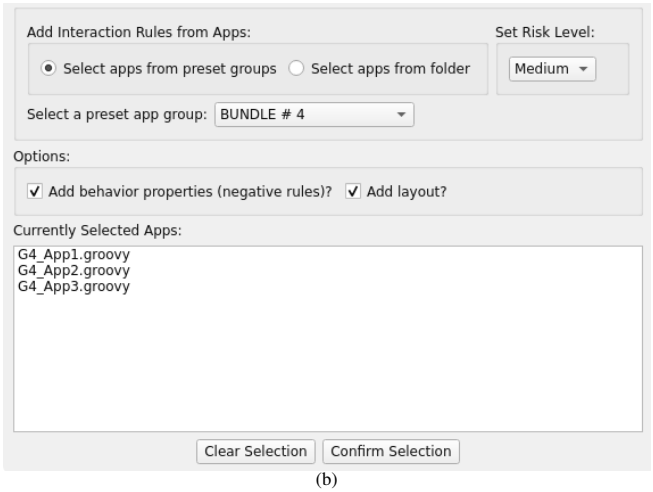
We compared InteractionShield against prior studies on rule conflict detection published over the past ten years. Table 15 summarizes the types of conflicts identified in each study, offering a comprehensive overview of existing approaches in this domain.

Table 16 and Table 17 provide detailed experimental results that support the observations reported in Table 6. We evaluated InteractionShield over two datasets: the first, previously used in IoTSan [21], Soteria [13], IoTCom [9], TAPInspector [6], and TAPFixer [20], and the second, adopted by HomeGuard [8]. The detected rule conflicts are summarized in Table 16 and Table 17, respectively.

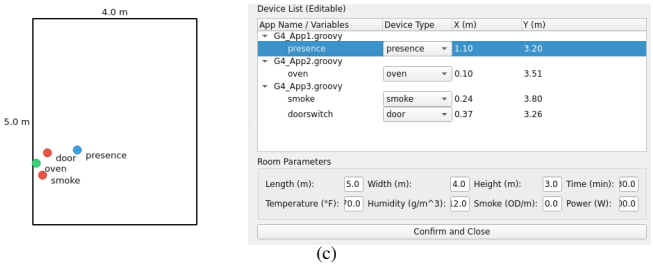
In both tables, conflicts with the subscript C denote *conditional* conflicts, while those with the subscript D denote *deterministic* conflicts. All possible conflicts are listed in the table. Conflicts shown in **bold** are detected exclusively by our approach. Conflicts that are underlined were detected by both prior work and our approach but are not considered conflicts under the current risk level. Conflicts in *italics* are detected only by our approach but are likewise not considered conflicts under the current risk level.

Figure 6: The GUI of InteractionShield: (a) App selector; (b) Device layout within the room; (c) Behavior properties input; (d) Threat detection results.

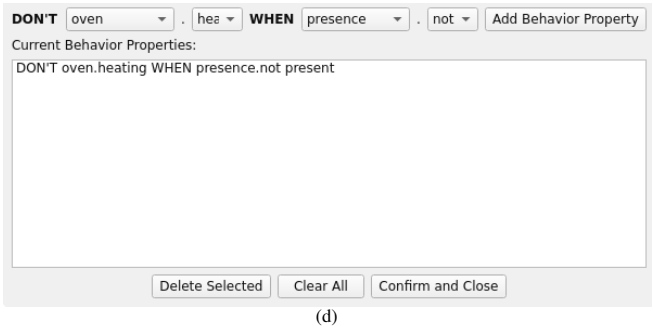
(a)



(b)



(c)



(d)

TABLE 15: Rule conflicts identified by existing detection approaches.

Papers	Year	Rule Conflicts									
		Duplication Action	Contradictory Action	Accumulation Impact	Reduction Impact	Force Action	Block Action	Disorder Action	Chain Enable	Chain Loop	Chain Disable
UTEA [39]	2014	●	◐	○	●	○	○	○	○	◐	○
CityGuard [15]	2017	○	●	●	●	○	○	●	○	○	○
Soteria [13]	2018	●	◐	○	○	○	○	○	○	○	○
IoTGuard [10]	2019	●	◐	○	○	○	○	○	○	◐	○
iRuler [11]	2019	●	◐	○	○	○	○	○	○	◐	◐
Brackenbury et al. [12]	2019	●	○	○	○	○	○	●	○	●	○
EUDebug [14]	2019	●	◐	○	○	○	○	○	○	◐	○
IoTC^2 [40]	2019	●	●	●	●	○	○	○	○	○	○
HomeGuard [8]	2020	●	◐	○	○	○	○	○	◐	◐	◐
IoTCom [9]	2020	●	●	○	○	○	○	○	●	◐	●
Ibrhim et al. [34]	2020	●	◐	○	●	○	○	○	○	◐	○
Mekuria et al. [41]	2020	●	●	○	●	○	○	○	○	◐	○
Huang et al. [31]	2021	○	◐	●	●	○	○	○	◐	○	○
TAPInspector [6]	2022	○	◐	○	●	○	●	●	◐	○	◐
Chi et al. [7]	2022	○	◐	○	○	○	○	●	●	◐	◐
IoTMediator [5]	2023	○	◐	○	○	○	○	○	●	◐	◐
TAPFixer [20]	2024	○	◐	○	○	○	●	●	◐	○	◐
CCDF-TAP [42]	2024	●	○	○	●	○	○	○	◐	◐	○
Han et al. [43]	2024	●	◐	○	●	○	○	○	◐	○	○
PSA [44]	2024	○	●	○	●	○	●	●	○	○	○
CP-IoT [35]	2024	●	●	○	○	○	○	○	●	●	○
InteractionShield	2025	●	●	●	●	●	●	●	●	●	●

Note: ●: included, ◐: partially included, ○: not included.

TABLE 16: App groups and rule conflicts in Table 6.

Group	App Names	Rule and Configuration	Rule Conflicts
#1	G1_App1 G1_App2	If the presence sensor detects presence, turn on the light; otherwise, turn off the light. When motion is detected, turn on the light, then turn it off after a delay.	C.1$_C$, C.2$_C$
#2	G2_App1 G2_App2	When the door is locked, set the mode to "Home". When switching to "Home" mode, turn off the appliance.	C.8$_C$
#3	G3_App1 G3_App2 G3_App3	If smoke is detected, turns on the light. When the light is turned on, set the mode to "Home". When the mode changes to "Home", lock the door.	C.8$_C$
#4	G4_App1 G4_App2 G4_App3	When the user is not present, change the location mode to "Home". When set to "Home" mode, turn on the oven. When smoke is detected, open the door.	**C.4**$_C$, C.8$_C$
#5	G5_App1 G5_App2	When motion is detected, turn on the light. When illuminance exceeds a certain threshold, turn off all lights.	**C.2**$_C$, C.8$_D$
#6	G6_App1 G6_App2	When the illuminance exceeds a certain level, turn off all lights. If the light is off, turn it on.	**C.2**$_C$, C.8$_C$, **C.8**$_D$, **C.9**$_C$
#7	G7_App1 G7_App2	When temperature rises above 30°C, turn off the heater; when temperature drops below 20°C, turn on the heater. If the power rises above the threshold, turn off the smart plug.	C.6$_C$, **C.7**$_C$, C.8$_D$, **C.9**$_D$, C.10$_C$
#8	G8_App1 G8_App2	If the user is present, turn on the heater; if the window is closed, turn off the heater. If the temperature rises above 28°C, open the window; if it drops below 15°C, close the window.	**C.3**$_D$, **C.4**$_C$, **C.8**$_D$, **C.9**$_D$, C.10$_D$

TABLE 17: App groups and rule conflicts in Table 7.

Group	App Names	Rule and Configuration	Rule Conflicts
#1	CurlingIron VirtualThermostat	When motion detected, turn on fan for 30 minutes. When motion detected, if temperature is lower than 72°F, turn off fan.	C.2$_D$, **C.8**$_D$
#2	NFCTagToggle LockItWhenILeave	When the user touches on mobile app, toggle switch and toggle door lock. When presence sensor becomes "not present", lock door.	**C.1**$_C$, C.2$_C$
#3	CurlingIron SwitchChangesMode MakeItSo	When motion detected, turn on oven. When oven is turned on, set home to "party" mode. When changed to "party" mode, unlock door and turn on heater.	**C.3**$_C$, C.8$_C$
#4	It'sTooHot EnergySaver	When temperature exceeds 80°F, turn on fan. When power usage exceeds 3000W, turn off fan.	C.2$_C$, C.8$_D$, **C.9**$_D$
#5	SmartHumidifier HumidityAlert	When humidity is below 30%, turn on humidifier; when humidity exceeds 50%, turn off humidifier. When humidity exceeds 50%, turn on vent; when humidity is below 30%, turn off vent.	**C.3**$_D$, C.8$_D$, C.9$_D$, *C.10*$_D$
#6	LightUptheNight	When illuminance exceeds 50 lx, turn off light; when illuminance gets below 30 lx, turn on light.	C.9$_D$
#7	BrightenDarkPlaces LetThereBeDark	When door is opened, if illuminance is below 10 lx, turn on light. When door is opened, turn off lights; when door is closed, restore the state of lights.	**C.1**$_D$, **C.2**$_D$, C.8$_D$, *C.10*$_C$
#8	ForgivingSecurity ScheduledModeChange	When motion sensor becomes "active", if the home is in "Away" mode, siren alarm. Set home to "Away" mode at 10 am and set home to "Night" mode at 6pm.	C.8$_C$, C.10$_C$
#9	ForgivingSecurity RiseAndShine	When motion sensor becomes "active", if home is in "Work" mode, turn on light after 1 second. When motion sensor becomes "active", set home to "At-Home" mode ("Work" mode).	C.10$_D$ (C.8$_D$)
#10	GoodNight OnceADay MakeItSo	When motion detected, if fan is off, set home to "sleep" mode. Turn on fan at 11 pm and turn off fan at 12 am. When changed to "sleep" mode, lock door.	*C.8*$_C$, C.10$_C$