Delay Wreaks Havoc on Your Smart Home: Delay-based Automation Interference Attacks

Haotian Chi[1]*, Chenglong Fu[1]*, Qiang Zeng[2], Xiaojiang Du[3]
[1] Department of Computer and Information Sciences, Temple University, Philadelphia, PA 19122, USA
[2] Department of Computer Science and Engineering, University of South Carolina, Columbia, SC 29201, USA
[3] Department of Electrical and Computer Engineering, Stevens Institute of Technology, Hoboken, NJ 07030, USA
* The first two authors contributed equally to this work.
Email: {htchi, chenglong.fu}@temple.edu, zeng1@cse.sc.edu, xdu16@stevens.edu

2022 IEEE Symposium on Security and Privacy (SP) | 978-1-6654-1316-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/SP46214.2022.9833620

***Abstract*—With the proliferation of Internet of Things (IoT) devices and platforms, it becomes a trend that IoT devices associated with different IoT platforms coexist in a smart home, demonstrating the following characteristics. First, a smart home may use more than one platform to support its devices and automation. Second, IoT devices of a home may transmit messages over different paths. By selectively delaying IoT messages, our study finds that two issues, *inconsistency* and *disorder*, can be exacerbated by attackers significantly. We then explore how these issues can be exploited and present seven types of exploitation, collectively referred to as *Delay-based Automation Interference* (DAI) attacks. DAI attacks cause home automation to yield incorrect interaction results, placing the IoT devices and smart home in insecure, unsafe, or unexpected states. It is worth highlighting that DAI attacks do not depend on any IoT implementation vulnerabilities or leaked keys/tokens, and they do not trigger alarms at any layers of the IoT protocol stack. To demonstrate and evaluate the new attacks, we set up two real-world testbeds, where commercial IoT devices and apps are deployed. The week-long experiments from both testbeds show that an attacker has adequate opportunities to launch DAI attacks that cause security or safety issues.**

I. Introduction

Rapid development of Internet of Things (IoT) has led to flourishing smart environments, (e.g., smart homes, offices, and laboratories). IoT platforms, such as Apple HomeKit [1], Samsung SmartThings [2], and Amazon Alexa [3], enable configurations of automation rules for interactions between IoT devices in a home, also known as *home automation*. For example, users can create *automations* on SmartThings, or *routines* on Alexa, to have their devices automatically react to sensor measurements, device status, time, etc.[1]

When multiple rules interplay in a physical environment, they may interfere with each other and cause unexpected automation. The cross-rule interference (CRI) problem has been intensively studied (on individual platforms) [4], [5], [6], [7], [8], [9], [10], [11], [12], [13]. However, existing work that studies the CRI problem makes the following two assumptions: (1) They assume that all rules run on the same platform [6], [7], [9], [10], [5], [12], [13], [8] (or rules on different platforms do not interact with one another and *can* be analyzed separately [11]); (2) They also assume that all IoT messages are transmitted with identical delays. For example, the first systematic categorization of CRI [4], [5] assumes both.

[1]We refer to automations and routines as *automation rules* or *rules*.

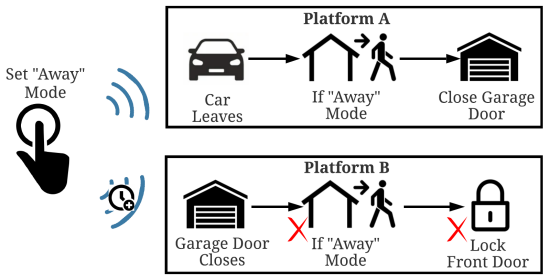


Fig. 1: An example showing a unique CRI problem. If the transmission of the *button-pressed* event (which sets "Away" mode) to Platform B suffers a non-negligible delay due to the DAI attack, the rule on Platform B will fail to lock the door.

The assumptions, however, do not necessarily hold true in real-world systems. Due to the fragmented IoT ecosystem [14], [15], different platforms are compatible with different subsets of IoT devices. When users cannot find a single platform to work with all their devices, they need to use multiple platforms. Furthermore, different devices use heterogeneous communication technologies and transmit messages through different paths. For example, a ZigBee device talks with a platform A's server via an IoT hub, while a WiFi-based device talks with its vendor's cloud B, which then delegates access to the device to the platform A. The communication paths are different and thus have different transmission delays. Worse, such delays can be manipulated by attackers without relying on any implementation vulnerabilities (Section II-C).

We thus consider a more general and realistic smart home system, where (1) users may use more than one platform to support their devices and install automation, and (2) IoT devices can transmit messages via more than one communication path. We classify smart home systems into three categories: single-platform single-path (SPSP) systems, single-platform multi-path (SPMP) systems, and multi-platform (MP) systems, which certainly contain multiple paths (Section III describes the three categories in details). By incorporating message transmission delays as a factor, we study two issues which do not exist in SPSP (where the two assumptions aforementioned hold): *disorder* and *inconsistency*. *Disorder* occurs when two

DOI 10.1109/SP46214.2022.00146

IoT events (or commands) arrive at platforms (or devices) in an order different from their actual occurrence order, while *inconsistency* arises when two platforms have inconsistent observations on the state of the same device.

We then re-examine cross-rule interference (CRI) in SPMP and MP systems, and reveal a new family of attacks that are overlooked by previous work, referred to as *Delay-based Automation Interference* (DAI) attacks, which exploit the interaction of automation rules in SPMP or MP systems. Fig. 1 illustrates an example. If the *button-pressed* event (which sets the home mode on both platforms to "`Away`") sent to Platform B is delayed by an attacker, the automation rule on Platform A succeeds in closing garage door when the homeowner's car leaves, but the rule on Platform B fails to lock the front door. We utilize two attack primitives [16], *selective event delaying* and *selective command delaying* (which leverage TCP hijacking attacks to significantly delay IoT events and commands without requiring session keys or triggering any alarms), as a building block to realize the DAI attacks. In short, an attacker exacerbates the inconsistency and disorder issues to launch DAI attacks, which exploit new CRI patterns and cannot be detected by existing work on CRI [6], [7], [8], [9], [10], [5], [11], [12], [13].

To demonstrate and evaluate the new attacks, we set up two real-world testbeds in two apartments. In the testbeds, we examine a variety of IoT devices, six cloud-based platforms and two local platforms. We validate all of the new attacks in the testbeds and verify that the attacks put the devices into insecure, unsafe or unexpected states. The one-week data from both testbeds also demonstrate that the attackers have adequate opportunities to launch the attacks. Finally, we discuss countermeasures. Our contributions are as follows:

- Given the fragmented IoT ecosystem, we present a much more general and realistic smart home model, where multiple IoT devices associated with multiple IoT platforms interact in the same physical space. Under this model, we study the *disorder* and *inconsistency* issues and leverage delay-attack primitives to exacerbate them.
- Compared to existing work on CRI [4], [5], we are the first to incorporate the *delay* factor into analyzing CRI problems. We are thus able to reveal new CRI patterns and build Delay-based Automation Interference attacks that are overlooked by existing work. These attacks do not rely on any leaked tokens or implementation vulnerabilities. Unlike jamming or discarding packets, the attacks do not trigger alarms at any layers of the IoT protocol stack.
- We evaluate the proposed attacks in two real-world testbeds, where commercial off-the-shelf (COTS) IoT devices and multiple popular platforms are used. It is demonstrated that the attacks can cause various problematic home automations and put the IoT devices and home in hazardous states.

II. Background and Attack Model

A. Smart Home Systems

In modern smart homes, various components (e.g., IoT devices, IoT hubs, home router, and IoT platforms) interact in the same physical space. We show the architecture of a modern smart home ecosystem in Fig. 2.

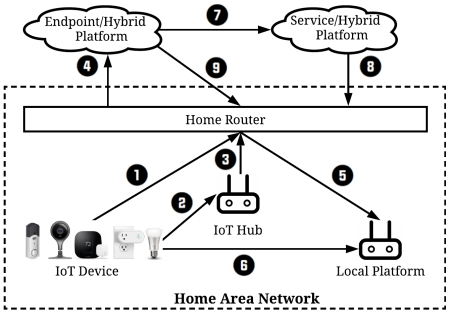


Fig. 2: Architecture of a smart home ecosystem. The arrows (labeled with circled numbers) denote the paths and directions of event flows. Command flows follow the same paths but in reverse directions.

Devices and Hubs: IoT devices consist of sensors and actuators. A sensor (e.g., temperature sensor) simply reports measurements of a physical property, while an actuator (e.g., smart lock) can receive and execute *commands* (e.g., unlock/lock) to change its status and report that after executing a command. The sensor measurements and actuator statuses are sent to an IoT platform via *events*. The current sensor measurement or actuator status is called the device's *state*. A device's state at a platform is updated by the device's latest event.

IoT devices employ a variety of techniques (e.g., WiFi, ZigBee, Z-Wave, Bluetooth) to send events or receive commands.[2] Devices that use WiFi (e.g., LIFX bulbs, Amazon Echo speakers) can connect to the home router directly ❶; those that use non-IP protocols (e.g., ZigBee, Z-Wave, Bluetooth, etc.) usually require a hub. The hub converts non-IP payloads from IoT devices ❷ to IP-based payloads, which can be sent to the home router ❸. The home router then forwards the IP-based payloads to cloud servers ❹ or to other devices in the local area network (LAN) ❺ for further processing. A local platform (e.g., HomeKit) can connect non-IP devices directly (❻). Multiple IoT hubs may be deployed in a home to accommodate their supported devices.

Platforms: Cloud-based IoT platforms may be classified into three types: endpoint, service, and hybrid. An endpoint platform is a messaging cloud that mediates communication between IoT devices and service platforms. It may be maintained by a device manufacturer or a third-party service provider (e.g., AWS IoT [17]). A service platform such as IFTTT (If This Then That) is a cloud that runs automation rules and usually obtains access to devices from endpoint platforms, via a cloud-to-cloud integration ❼. A hybrid platform is a combination of an endpoint and a service platform. Many popular smart home platforms, such as Amazon Alexa and SmartThings, belong to this category. In contrast to the aforementioned cloud-based platforms, a local platform (e.g., HomeKit, openHAB) is usually hosted on a local device (e.g., PC, laptop, HomePod speaker) in the home area network, and

[2]For the sake of brevity, we mainly discuss event flows. Command flows follow the same paths but in the reverse direction.

can connect with IoT devices directly, via non-IP protocols (e.g., ZigBee, Z-Wave, Bluetooth) ❻ or via LAN ❶→❺. A local platform can also access devices that have been connected to an IoT hub (❷→❸→❺) or an endpoint/hybrid platform (❶→❹→❾→❺ or ❶→❹→❼→❽→❺), by using the APIs provided by the hub (e.g., Hue bridge) or the endpoint/hybrid platform (e.g., LIFX cloud), respectively. For the sake of brevity, in this paper, we collectively use the term *platforms* to denote all service, hybrid, and local platforms that can run automation rules.

Automation: An automation rule is a reactive app that follows a *trigger-condition-action* paradigm. A rule's *trigger* specifies a constraint that a certain type of event (termed as *trigger event*) must satisfy to activate the rule. Before an action in a rule is taken, its *condition* is checked to verify whether states (e.g., devices states, time) have satisfied the predefined constraints. Different platforms may have distinct supports for defining rule conditions. For instance, SmartThings allows using both device states and time to define rule conditions, Philips Hue only allows the usage of time in conditions, and Amazon Alexa only supports trigger-action rules.

B. Inferring Home Configuration from Encrypted Traffic

Recent work has illustrated the effectiveness of inferring smart home configuration information (i.e., device types, automation apps, the app-device bindings, etc.) from encrypted network traffic. Side-channel attacks can utilize the metadata in the traffic, such as source/destination IP/MAC addresses, DNS, packet lengths, frequencies, etc., to identify device information (e.g., manufacturer, model) [18], [19], [20], [21], [22], [23], [24] and recognize events/commands in real time [25], [26], [27], [28], [29], [30]. By analyzing a sequence of accurately-recognized IoT events/commands, routines and automation rules can also be inferred [26], [31], [32], [30]. For instance, the device identification [18] and [19] achieve an accuracy of 0.91 and 0.81, respectively. The average accuracy for inferring events [29] is 0.97. The precision and F1 score for inferring automation rules [26] are 1 and 0.96, respectively. This work utilizes side-channel attacks to infer smart home configuration information and build attacks.

C. Selective Event/Command Delaying

In smart home systems, most communication paths (see Fig. 2) between an IoT device/hub and a platform go through a home router.[3] On IP/TCP links, events and commands are typically transmitted using the SSL/TLS protocol, which runs on top of the transport layer. Each pair of an IoT device/hub and a cloud establishes a unique TLS session. IoT events and commands are conveyed in a specific type of TLS record, i.e., Application record, whose *type* field in header is "0x17".

Although TLS provides the confidentiality and order-preserving features, neither TLS nor the upper application-layer protocols used by smart home systems, such as HTTP and MQTT, have a strict liveness checking on messages. IoT devices and clouds usually exchange TLS-protected heartbeat (a.k.a., keep-alive) messages periodically. If an IoT device/-cloud cannot receive a heartbeart request/reply or an event/-command ack from the other side within a pre-defined time period (usually tens of seconds; see Section V-B), i.e., a *timeout* occurs, it will actively disconnect the TCP connection and try to reconnect. If an attacker succeeds in performing a TCP hijacking attack (see Section II-D) between an IoT device/hub and a platform, he can establish a TCP connection with each side, becoming a *relay node* in the middle. Although the MITM attacker cannot decrypt TLS-protected messages, he can delay forwarding the messages. Delaying messages cannot be detected by the IoT protocol stack as long as it does not trigger a timeout. An attacker can recognize IoT events and commands from the encrypted packets through side-channel analysis (Section II-B) and selectively delay a specific event or command, which is referred to as two attack primitives: *selective event delaying* and *selective command delaying*. Our prior work [16] discussed the two attack primitives in detail. In this paper, we use the two primitives as a building block.

D. Attack Model

Who Can Launch the Attacks? We are concerned with an attacker who can eavesdrop and delay the encrypted traffic between IoT devices/hubs and the IoT cloud/local platforms. For example, he can compromise the WiFi router in the victim home, or perform ARP spoofing attacks [33], [34] from a local IoT device (e.g., compromised by Mirai attacks [35], [36]). When the attacker and the victim share a WiFi (e.g., at a company, hospital, or facility campus), the attacker can launch sniffing and ARP spoofing from his own device conveniently. If an attacker has compromised an ISP router, he can launch attacks at scale against many homes that use cloud-based platforms. Through the attack, the attacker obtains two capabilities: (1) passively analyzing traffic; and (2) actively delaying events/commands.

Passive Observation. For this purpose, sniffing attacks are sufficient (i.e., TCP hijacking is not needed). Specifically, the attacker has access to headers of the data link, network and transport layers (such as device MAC addresses, IP addresses and ports), and the type and length of TLS records. The attacker utilizes the techniques discussed in Section II-B to obtain knowledge of the victim home and recognize events/-commands from the traffic.

Active Delaying. The attacker selectively delays events/commands transmitted over a hijacked TCP session. To evade detection, the attacker does *not* discard any events/commands or delay them for an excessive period. The delay range (without being detected) depends on the type of IoT devices and platforms (more details are given in Table IV).

III. Event/Command Disorder and Inconsistency

In a smart home where all IoT devices use the same TCP/IP communication path to exchange events and commands with a single platform, as shown in Fig. 3, two properties hold:

[3] An exception is the communication between non-IP devices and a local platform (❻).

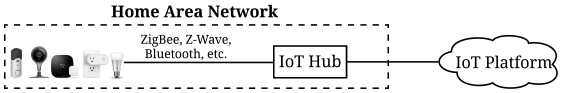


Fig. 3: Single-platform single-path device connection.

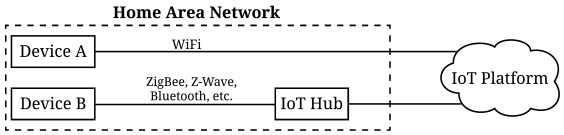


Fig. 4: Single-platform multi-path device connection.

- **Same Order.** All events arrive at the IoT platform in the same order as they arrived at the hub. This holds for all events that are transmitted on the same TLS session (i.e., the hub-platform path), as each TLS record includes a Message Authentication Code that checks data integrity and the sequence number [37]. Likewise, a command sent earlier by the platform arrives at the hub earlier.
- **Consistency.** The executions of all automation rules are based on a consistent observation about the smart home, because the same database maintained by the platform is queried for data (e.g., device states, home mode, etc.).

Although existing research on CRI [38], [39], [6], [7], [8], [26], [9], [40], [5] considers or implicitly assumes the single-platform single-path (SPSP) deployment model, it does not reflect the reality. Over time, users may purchase a variety of IoT devices and connect them with multiple platforms. Therefore, the following two deployment models are common in reality: the single-platform multi-path (SPMP) model and the multi-platform (MP) model. According to our online survey (see Appendix A for details) including 85 realistic smart homes, SPSP, SPMP and MP deployments account for 17.6%, 20.0% and 62.4%, respectively.

A. Single-Platform Multi-Path: Disorder

When IoT devices use multiple TCP/IP paths to exchange messages with a platform, the *same order* property will not hold true. Fig. 4 shows an example. Assuming that device *A*'s transmission path (i.e., the TLS between device *A* and the platform) is delayed, even if events from *A* are generated earlier, its events may arrive at the platform later than those of device *B*. We refer to this issue as a *disorder*. Note that this issue could also happen to commands.

B. Multi-Platform: Disorder and Inconsistency

In multi-platform systems, there exist multiple TCP/IP paths between IoT devices and platforms. Therefore, the *disorder* issue certainly exists. Plus, when an IoT device is connected to two or more platforms, the platforms may have different observations on the same device's state. This is because a new event from the device may have different delays when transmitted to the platforms (via different paths). As shown in Fig. 5, a WeMo smart plug communicates with the WeMo cloud (a.k.a., an endpoint cloud) through the Internet, and talks in the local area network with a HomePod that hosts HomeKit or with a SmartThings hub which forwards communication

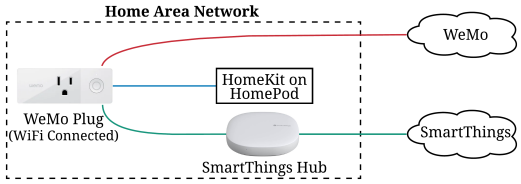


Fig. 5: WeMo smart plug connected to three platforms: WeMo, SmartThings, and HomeKit.

to the SmartThings cloud. Thus, the WeMo smart plug uses three different paths to connect with the three platforms. The transmission delays could create a time window, during which platforms have inconsistent observations on the state of the plug (i.e., ON/OFF). We refer to this issue as *inconsistency*.

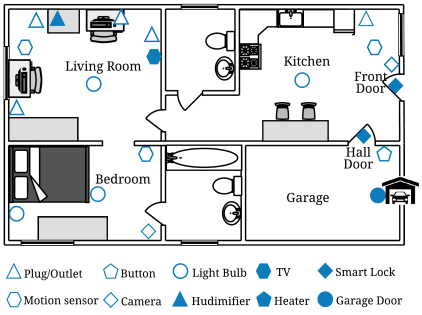
In non-adversarial scenarios where the network delays are usually small, e.g., less than one second, the disorder and inconsistency issues are not severe. However, in the presence of an attacker who intentionally delays events/commands, the disorder and inconsistency issues may be manipulated by the attacker to cause serious security and safety threats to a smart home (e.g., leaving the front door unlocked when the owners are not home), and this is discussed in Section IV.

IV. Delay-based Automation Interference Attacks

We consider an attacker who launches selective event/command delaying attacks to cause disorder and inconsistency issues in order to interfere with home automation, leading to incorrect, unexpected, and hazardous automation. These attacks are collectively referred to as Delay-based Automation Interference (DAI) attacks. Different from the well-studied cross-rule interference problems (e.g., [5], [6], [7], [9]) due to mis-programming or mis-configuration, DAI attacks exploit CRI problems in SPMP and MP systems that cannot be detected by existing work. To systematically study and categorize of DAI attacks, we use a formal approach *process calculus* [41], [42] to model smart home deployments and extend a notion of *observation equivalence* for identifying CRI problems in a smart home. With the theoretic basis, we parameterize the configurations (including the message-transmission delay) of a smart home system and enumerate the possible configurations as well as the attacker's strategies to find all possible DAI attacks. Due to the page limits, we briefly present the basic idea of the utilization of observation equivalence in Section IV-A, and defer the complete formalization part to Appendices B and C. The rest of this section is focused on presenting the discovered attacks.

A. CRI Resistance Modeling

A smart home's physical environment is denoted as $\mathtt{E}$, and it has two automation rules $\mathtt{R1}$ and $\mathtt{R2}$ that run on two platforms L_1 and L_2, respectively (L_1 and L_2 may refer to the same or different platforms). We use $\mathtt{Sys} = \mathtt{E} \circ (\mathtt{R1}[D_1 \triangleright L_1] \parallel \mathtt{R2}[D_2 \triangleright L_2])$ to denote this smart home system, where D_1 and D_2 are the sets of devices involved in $\mathtt{R1}$ and $\mathtt{R2}$, respectively. Communication paths between devices and platforms in $\mathtt{Sys}$ suffer from delay attacks. Suppose a *specification* system



Room	RID	Rule description (Format: When [trigger], if [condition], [action])
Garage	1	When the hall door is closed, if *away* mode, close garage door and set *home* mode. (arriving)
	2	When the *away* button is pressed, set to *away* mode. (leaving)
	3	When the hall door is closed, if *home* mode, open the garage door. (leaving)
	4	When the car leaves, if *away* mode, close the garage door.
	5	When the garage door is closed, if *away* mode, lock the front door and hall door.
Kitchen	6	When the user arrives, unlock the front door.
	7	When the front door is unlocked, turn on the indoor camera.
	8	When the front door is opened, if *away* mode, turn on all smart plugs.
	9	When cook time (12pm, 7pm), if motion is active, turn on the heater in the kitchen.
	10	When power high, if motion is inactive, turn off smart plug (that connects kettle, heater, etc.).
	11	When door is unlocked, if the user is away, sound the alarm and call the police.
Living Room	12	When motion is detected, turn on the humidifier and TV.
	13	When the luminance drops below 20 lux, if motion is active, turn on the lights.
	14	When the luminance exceeds 20 lux, if motion is inactive, turn off the lights.
Bedroom	15	When motion is detected, if user is at home, turn off camera.
	16	When motion is detected, if user is away, ring alarm and send live video to security company.
	17	When motion is detected, turn on the ceiling lamp.
	18	When motion is detected, if luminance is below 20 lux, turn on the floor lamp.

Fig. 6: A smart home illustrating part of the deployed IoT devices and automation rules.

$\texttt{Sys}^* = \texttt{E} \circ (\texttt{R1}[D_1 \triangleright L^*] \,\|\, \texttt{R2}[D_2 \triangleright L^*])$, where all devices $D_1 \cup D_2$ are connected to an oracle platform L^*, both rules R1 and R2 run on L^*, and all its communication paths incur identical delays. That is, $\texttt{Sys}^*$ runs the same rules and devices in the same environment as Sys but suffers no DAI attacks.

Observation equivalence is a property that two or more concurrent systems are indistinguishable regarding their observable implications (e.g., the states of sensors and actuators). Therefore, if the real system Sys (with DAI attacks) and specification system $\texttt{Sys}^*$ (without DAI attacks) are observationally equivalent while they evolve, i.e., the automation results (resultant device states) are always the same no matter how rules are triggered the same way in both systems, we say that the two automation rules R1 and R2 in the deployment Sys are *CRI-resistant* to DAI attacks. With this notion, we not only can verify if rules are CRI-resistant after formalizing a given smart home deployment, but also find all possible types of DAI attacks by traversing different attack strategies (i.e., which communication paths to delay). Due to the page limit, we defer full details of the formal modeling, and the methodology for observation equivalence analysis and DAI attack categorization, to Appendices B and C, respectively.

B. An Example Smart Home

To help present DAI attacks, we first describe an example smart home with multiple IoT devices and automation rules deployed, as shown in Fig. 6. Regarding Rules 1-3, note that whether a person enters or leaves, the resulting door event sequence is the same (e.g., "*unlocked* → *open* → *closed* → *locked*"), and therefore cannot be used to infer whether the homeowner enters or leaves the home; to distinguish arriving/leaving behaviors, a *mode* with possible values, such as *home* and *away*, can be set by the user manually (like using a mobile companion app or an ADT system [43]) or automatically (based on a presence sensor or automation), which then can be used to, e.g., open/close the garage door correctly. The example will be used to present the new DAI attacks in a more concrete fashion.

C. DAI Attacks

We summarize seven types of DAI attacks in Table I. For each type, Table I lists the attack name, the section interpreting it, the rule pattern describing the attack, the message(s) that should be delayed, the issue exploited and the consequence.

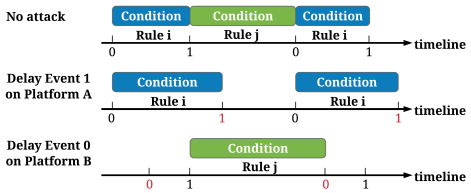


Fig. 7: Condition overlapping attack. The conditions of Rules i and j are satisfied when the device state is 0 and 1, respectively. Here, 0 and 1 broadly denote the values of a binary attribute, e.g., *inactive* and *active* of a motion sensor.

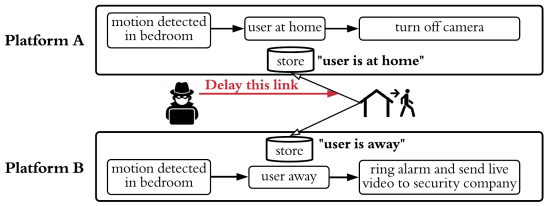


Fig. 8: A scenario of condition overlapping attack (action conflict).

1) Condition Overlapping Attack: Existing works [6], [7], [9], [10], [5], [11], [12], [13] assume that rules do not run simultaneously if their conditions are *exclusive*, (e.g., "*if motion sensor is active*" vs. "*if motion sensor is inactive*"). As shown in Fig. 7, suppose the conditions of Rules *i* and *j* check the state of a device, and this device has two possible states 0 and 1. When there are no attacks, the condition-satisfaction period during which Rule i's condition (e.g., "*if the state is 0*") is satisfied has *no* overlap with that of Rule *j*, (e.g., "*if the state is 1*"). However, if Rules i and j run on two different platforms A and B, respectively, by delaying the arrival of event 1 on platform A and/or the arrival of event 0 on platform B (see Fig. 7), the condition-satisfaction periods of the two rules will have overlaps. As a result, if both rules are triggered during the overlapping period, they will be executed during the same period, violating the expectation that they are mutually exclusive. We collectively define *condition-overlapping attacks* (COA) as attacks that exploit inconsistency issues to cause rules with exclusive conditions to run simultaneously. Under this attack, rules that are considered interference-free by prior

TABLE I: Summary of DAI attacks. The DAI attacks are derived from a systematic categorization show in Appendix C. The **Rule Pattern** column shows the pattern of victim rules targeted by each DAI attack. $R_i = (T_i, C_i, A_i), i = 1, 2$ denotes two victim rules, where T_i, C_i, A_i are the trigger, condition, and action, respectively. $\perp$ denotes "*mutually exclusive*"; $\wedge$ denotes "*overlaps*"; $\neg$ denotes that "*mutually contradictory*"; $\Rightarrow$ and $\not\Rightarrow$ denotes "*enables*" and "*disables*", respectively; $<\!\approx$ denotes "*is close but precedent to*"; $\dashv$ denotes "*requires*"; $\uplus$ denotes "*guarded by*".

Attack	Section #	Rule Pattern	What to Delay?[1]	Issue Exploited	Consequence
(1) Condition Overlapping Attack (COA)	IV-C1				
(1.1) COA - Action Conflict		$T_1 = T_2, C_1 \perp C_2, A_1 = \neg A_2$	$C_1 \rightarrow R_2$ ($C_2 \rightarrow R_1$)	Inconsistency	Action A_1 (A_2) Nullified
(1.2) COA - Infinite Loop		$A_1 \Rightarrow T_2, A_2 \Rightarrow T_1, C_1 \perp C_2$	$C_1 \rightarrow R_2$ ($C_2 \rightarrow R_1$)	Inconsistency	Infinite Loop
(1.3) COA - Chained Execution		$A_1 \Rightarrow T_2, C_1 \perp C_2$	$C_1 \rightarrow R_2$	Inconsistency	Chained Execution
(2) Trigger-Cond. Overlapping Attack	IV-C2	$A_1 \Rightarrow T_2, T_1 \perp C_2$	$T_1 \rightarrow R_2$	Inconsistency	Chained Execution
(3) Condition Diverging Attack (CDA)	IV-C3				
(3.1) CDA - Disabled Parallel Execution		$T_1 = T_2, C_1 = C_2$	$C_1 \rightarrow R_2$	Inconsistency	Action A_2 Suppressed
(3.2) CDA - Disabled Chained Execution		$A_1 \Rightarrow T_2, C_1 = C_2$	$C_1 \rightarrow R_2$	Inconsistency	Action A_2 Suppressed
(4) Action Disordering Attack	IV-C4	$T_1 <\!\approx T_2, C_1 \wedge C_2, A_2 \dashv A_1$	$T_1 \rightarrow R_1$ and/or A_1	Disorder	Actions Disordered
(5) Condition Disabling Attack	IV-C5	$T_1 = T_2, A_1 \not\Rightarrow C_2$	$T_2 \rightarrow R_2$	Disorder	Action A_2 Suppressed
(6) Condition Enabling Attack	IV-C6	$T_1 = T_2, A_1 \Rightarrow C_2$	$T_2 \rightarrow R_2$	Disorder	Unexpected Action A_2
(7) Action Delaying Attack (ADA)	IV-C7				
(7.1) ADA - Delayed Parallel Execution		$T_1 = T_2, C_1 \wedge C_2, A_1 \uplus A_2$	$T_2 \rightarrow R_2$ or A_2	Delay	Action A_2 lags behind A_1
(7.2) ADA - Delayed Chained Execution		$A_1 \Rightarrow T_2, C_1 \wedge C_2, A_1 \uplus A_2$	$T_2 \rightarrow R_2$ or A_2	Delay	Action A_2 lags behind A_1

[1] Delaying a trigger $T_{(\cdot)}$ or condition $C_{(\cdot)}$ to a destined rule $R_{(\cdot)}$ is achieved by delaying the event, checked by $T_{(\cdot)}$ or $C_{(\cdot)}$, to the platform where $R_{(\cdot)}$ runs. For instance, $C_1 \rightarrow R_2$ denotes that an event (e.g., *motion active*) which makes the condition C_1 (e.g., "*if the motion is detected*") true should be delayed when it is transmitted to the platform hosting the rule R_2. Delaying an action $A_{(\cdot)}$ is realized by delaying the command issued by $A_{(\cdot)}$ when the command is sent to the destined device.

work become problematic. Below, we present three sub-types of the attack and use concrete examples to demonstrate how they lead to incorrect automation results.

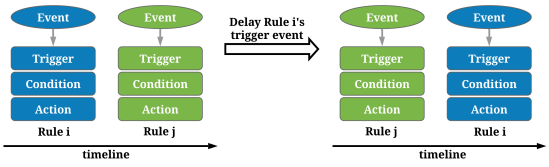
Action Conflict. Through the condition overlapping attack, an attacker can cause an *action conflict*. Consider Rules 15 and 16 (in Fig. 6) which protect user's privacy and detect burglary respectively, based on the user's presence when motion is detected in the bedroom. As shown in Fig. 8, if they are installed on different platforms A and B, the attacker can delay the user-away event from a presence sensor to platform A. Thus, when triggered by the motion-active event, Rules 15 and 16 have different observations on the user's presence: "*user is at home*" and "*user is away*", from the databases of platforms A and B, respectively, and perform conflicting actions. Consequently, Rule 15 turns off the camera, preventing Rule 16 from recording live videos.

Infinite Loop. Rules 13 and 14 are configured to control lights in the living room, based on luminance and user motion. According to the recent research [6], [5], [9], the two rules have no interference, since their conditions, "*if motion is active*" and "*if motion is inactive*", are mutually exclusive. Specifically, when Rule 13 turns on the lights and brightens the room (which means the motion is active), Rule 14 is activated but its condition is not satisfied; thus, Rules 13 and 14 cannot trigger each other according to existing work. However, when the rules run on two different platforms, the condition overlapping attack can delay the most recent motion-active event sent to the platform hosting Rule 14. As a result, when Rule 13 turns on the lights, Rule 14 will be triggered to turn them off, which triggers Rule 13 once again; during the period the motion-active event is delayed, an *infinite loop* will make the lights continuously flash on and off.

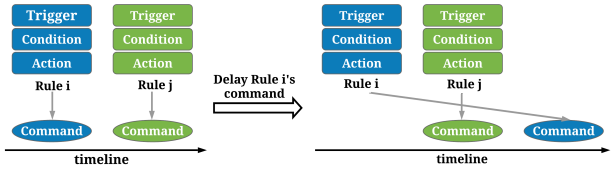
Chained Execution. *Chained execution* of rules [5] (also referred to as interaction chain [40], rule chains [44], or feature chaining [45]) is a well-studied CRI pattern, which occurs when the action of one rule activates the trigger of another rule. Let's consider Rule 9 in Figure 6, which turns on an Internet-connected space heater (such as [46]) during cooking time (12pm and 7pm). For energy-saving and safety reasons, Rule 10 turns off the smart plug when appliances connected to it increase power consumption over a threshold if nobody is in the kitchen. Rule 9 uses a condition, "*if motion is active*", to *avoid* chaining with Rule 10, whose condition is "*if motion is inactive*". Without attacks, the coexistence of Rules 9 and 10 will not cause chained execution. However, if Rules 9 and 10 run on different platforms, A and B, respectively, the attacker can delay a motion-active event being sent to platform B, causing Rule 10's condition to be kept true. This way, when Rule 9 is activated at cooking time, an unexpected chained execution of Rule 10 occurs to turn off the smart plug.

2) Trigger-Condition Overlapping Attack: Some automation rules avoid chained execution by making the trigger of one rule and the condition of another mutually exclusive. However, the exclusiveness is broken by delaying a trigger event. Consider Rules 6 and 11 in Fig. 6. Rule 6 unlocks the front door when the user arrives home (detected by a presence sensor); Rule 11 automatically sounds an alarm and call the police when it detects a possible break-in: the front door is unlocked while the user is not present. Rules 6 and 11 play their own roles and do not chain when they have consistent observations on the presence sensor. Suppose Rules 6 and 11 run on two platforms, A and B, respectively. If a user-present event sent to platform B is delayed by the attacker, Rule 11 uses outdated information (i.e., the user is not present) to evaluate its condition when it is triggered by the door-unlocked event (caused by Rule 6's action). As a result, it causes a false burglar alarm.

3) Condition Diverging Attack: In contrast to the condition overlapping attack, which causes rules with exclusive conditions to interact, the *condition diverging attack* prevents rules with overlapping rules from interacting. Although chained execution is recognized as a CRI pattern, it is sometimes an important feature for grouping rules for specific goals if it is utilized properly. For example, Rules 4 and 5 use this feature to



(a) By delaying trigger event



(b) By delaying command

Fig. 9: Action disordering attack.

close the garage door and lock the doors in a row, which do not cause any problem if they run on the same platform. However, the condition diverging attack can disable the desired chained execution. Suppose Rules 4 and 5 run on different platforms, A and B, respectively. The attack can desynchronize the mode on platforms A and B by delaying the "*away*" message sent to B. Thus, when the user drives away, Rule 4 is executed to close the garage door, while Rule 5's condition is not satisfied, failing to lock the front and hall doors.

4) Action Disordering Attack: This attack changes the arrival order of commands issued by different rules. Consider Rules 8 and 12 in Fig. 6: Normally, a user first enters her home through the door and then her motion in the living room is detected; as a result, Rule 8 is first triggered by the door-open event to turn on smart plugs, and then Rule 12 is triggered, which turns on the humidifier and TV. This order is presumably ensured by the order of physical activities. However, if the command due to Rule 8 is delayed by attacks, Rule 12 will fail to turn on the humidifier and TV.

More generally, an action disordering attack can be achieved by delaying the trigger event, command, or both. Assume that Rule i is supposed to take its action before Rule j (assume neither rule involves the use of timers). As shown in Fig. 9(a), when Rule i's trigger event is delayed for a sufficient period, Rule i is executed after Rule j. Fig. 9(b) shows another attack strategy: an attacker can delay the command of Rule i and make it arrive at the destined device later than the command of Rule j, even though Rule i executes before Rule j. Alternatively, the attacker can delay both the trigger event and command of Rule i to increase the total delay.

5) Condition Disabling Attack: If a rule's action changes a device's state to a value that dissatisfies the condition of another rule, it is called *Condition Block* [9] CRI. Prior work [5] empirically demonstrates that, if the two rules subscribe to the same trigger event and run on the same platform, *Condition Block* is unlikely to happen. However, as shown in Fig. 10, when the two rules are on different platforms, a condition disabling attack becomes possible, because the attacker can delay the trigger event sent to the platform hosting Rule i, such that Rule j changes the device state (green), which disables

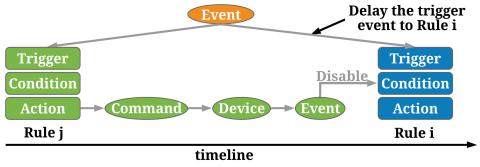


Fig. 10: Condition disabling attack.

the condition of Rule i.

Let's consider Rules 17 and 18. When a user enters the bedroom, Rule 17 on platform A turns on the ceiling lamp; if the room is too dark, Rule 18 turns on the floor lamp as well. If Rule 18 is also installed on platform A, the two rules ensure that when the user enters the bedroom, either (1) only the ceiling lamp is turned on if initially there is at least 20 lux luminance, or (2) both lamps are turned on if luminance is below 20 lux. However, as platform A (such as Amazon Alexa) does not support defining rule conditions, Rule 18 is installed on another platform B. The automation now becomes vulnerable to the condition disabling attack. An attacker can delay the motion-active event to platform B. Thus, when motion is detected, Rule 17 first turns on the ceiling lamp, increasing the measurement of the luminance sensor to a higher value, say 22 lux. The new luminance value will then be sent to platform B. When the delay attack ends, Platform B receives the motion-active event and Rule 18 is triggered. As Rule 18 now observes the new illuminance value, the condition disabling attack disables Rule 18 to take its action.

6) Condition Enabling Attack: This attack materializes another CRI pattern *Condition Bypass* which happens when Rule j's action changes a device's state to one that satisfies Rule i's condition. Similar to *Condition Block*, *Condition Bypass* can hardly happen when Rules i and j run the same platform, but becomes possible if they are on different platforms.

For example, Rules 1 and 3 in Fig. 6, which are used for controlling the garage door. Suppose the home mode is "*away*" when both rules are triggered by the hall door-closed event. If they run on the same platform, it is impossible for Rule 1's action (set to "*home*" mode) to have any impact on Rule 3's condition, since Rules 1 and 3 are triggered simultaneously. However, if Rules 1 and 3 run on different platforms A and B, respectively, the condition-enabling attack can be launched by selectively delaying the door-closed event to the platform B. When Rule 3 receives the delayed door-closed event, the mode has been set to "*home*" on platform B. As a result, Rule 3 will pass its condition checking and leave the garage door open, which causes safety issues.

7) Action Delaying Attack: Some rules are configured to run together for certain purposes, i.e., they are usually triggered in parallel by the same event, or in a row through a chained execution, and one rule's action can safeguard another. For instance, a user installs Rule 6 (see Fig. 6) to unlock the garage door when he arrives home. However, he is concerned that this rule is unsafe because the presence sensor (based on the location of a smartphone or specialized device) usually has a low precision, i.e., Rule 6 may unlock the door even when the user just passes by or is approaching but still far away

from his home, leaving a chance for burglary. To secure Rule 6, Rule 7 is installed to monitor the door with a surveillance camera when the door is unlocked. Rules 6 and 7 run one after another because the action of Rule 6 triggers Rule 7. However, the action delaying attack can delay the action of turning on the camera in Rule 7 for a sufficiently-long period, such that Rule 7 fails to secure Rule 6. Similar to an action disordering attack, an action delaying attack can be realized by delaying the target rule's trigger event, command, or both.

D. Factors Affecting Attack Successes

Rules are *vulnerable* to a certain type of attack if they satisfy the corresponding **Rule Pattern** and **Deployment Model** (see Table I). An attacker launches attacks by delaying the events/commands as shown in **What to Delay?** (in Table I).

Given a pair of vulnerable rules and the applicable attack type, whether an attack can succeed depends on two factors: (1) allowed message delay length, and (2) user behaviors. A larger delay gives a wider time window for delaying a trigger event/a command that enables or disables a rule condition. The allowed delay length, denoted as $\Delta T_{allowed}$, depends on the IoT device and platform (see the **Delay Range** testing in Section V-B1).

User behaviors affect the order and time interval the vulnerable rules are triggered. Consider the *action disordering attack* on Rules 8 and 12 in Section IV-C4 as an example. Rules 8 and 12 are triggered by two user activities: opening the front door and entering the living room, which generate *door-open* and *motion-active* events, respectively. The interval between the two activities is denoted as $\Delta T_{interval}$. If the user approaches the motion sensor in the living room within $\Delta T_{allowed}$ after she opens the front door (i.e., $\Delta T_{allowed} > \Delta T_{interval}$; note that we have ignored the difference between the transmission time for sending the two trigger events, as it is usually negligible when there are no attacks), Rule 12 will be triggered earlier than Rule 8 (i.e., the execution order is reversed), leading to a successful attack. Otherwise, the attack fails.

While user behaviors change time by time, there usually exists a pattern and the pattern can be learned from historical events and commands, which can be inferred using side channel attacks [27], [30], [31] (see Section II-B). If the attacker finds that the user never goes to the living room within $\Delta T_{allowed}$ after she opens the front door (i.e., $\Delta T_{allowed} < \Delta T_{interval}$), he chooses not to perform an *action disordering attack* on Rules 8 and 12 since it will never succeed. If $\Delta T_{interval}$ is smaller than $\Delta T_{allowed}$ with a high probability, then the attack success rate is also high. Note that failed DAI attack attempts remain stealthy since the delay does not trigger alarms at any layers of the IoT protocol stack.

V. Evaluation

In Section V-A, we describe the deployment details of two real-world smart home testbeds used for evaluating DAI. In Section V-B, we validate DAI attacks in the two testbeds. In Section V-C, we evaluate the attack opportunities and success rates of DAI attacks on a daily life basis.

A. Smart Home Testbeds and Attack Implementation

There are no public datasets of smart homes (including devices, rule sets, and configuration). Thus, like previous work on IoT security research [8], [26], [47], we set up smart home testbeds, denoted as T_1 and T_2, which are in two real homes to evaluate the DAI attacks. We received the IRB approval (see Appendix D for details). There are two persons (a male graduate student and a female graduate student in their 30s and 29s, respectively) living in testbed T_1, and one person in T_2 (a 27-year-old male graduate student). None of the testbed members are the authors. The smart home layouts and the IoT devices in each smart home are given in Fig. 11 and Table II, respectively. In total, 36 automation rules are installed on 4 automation platforms to interact with 55 IoT devices. The automation rules, which are listed in Table III, are chosen from the official app stores [48] or open-source datasets [49], and the final configurations are based on the discussion between the researchers and the residents living in the testbeds. Each testbed has a WiFi router, providing a WiFi access point and a few Ethernet ports for the deployed IoT devices.

A Raspberry Pi 4 Model B with a 2GB RAM and a 32GB MicroSD card is placed in each testbed to simulate a device compromised by the attacker. It is worth noting that if the attacker and the victim share a WiFi (e.g., at a factory, company, hospital, or university), or the attacker has stolen the WiFi password, the attacker can launch the attacks directly from his device. Plus, an attacker who has compromised the smart home router or has physical contact with the cable can also launch attacks without relying on ARP spoofing. Note ARP spoofing is decades-old mature attacks for hijacking traffic. A famous tool, IoT Inspector [50], has demonstrated that ARP spoofing can hijack a large amount of IoT traffic without causing network instability. We use the ARP spoofing-based technique to turn the Raspberry Pi into a relay node that can examine and delay the traffic between IoT devices/hubs and the WiFi router, or between IoT devices and hubs. By configuring the firewall rules through `iptables`, all traffic forwarded by the Raspberry Pi is under the control of our attack script. The attack script uses the approach in [29] to recognize events/commands from encrypted traffic. The approach [29] constructs packet-level signatures of IoT events and commands based on source & destination IPs and payload lengths in an offline phase, and then detects events and commands with the signatures in runtime with an accuracy over 97%. The attack script utilizes a DFA matching approach [26] to infer automation rules from the event and command logs of a couple of days (one week in our experiment), achieving an accuracy of 94%. DAI attacks are performed based on the inferred home configuration.

B. Validation of Attacks

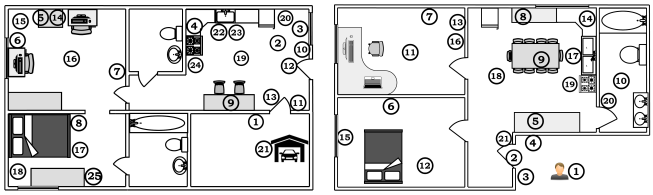
We present the methodology and results for validating the DAI attacks in the two testbeds.

1) Methodology: To ease the validation, we ask the testbed members to assist in triggering the automation rules by behaving in controlled patterns. The given instructions incorporate

TABLE II: IoT devices and their connections to platforms in T_1 and T_2. **d-ID**: device ID. Acronyms: SmartThings (ST), Philips Hue (PH).

Testbed T_1			Testbed T_2		
d-ID	**Device**	**Connection Path to Platform**[1]	**d-ID**	**Device**	**Connection Path to Platform**
–	SmartThings hub	• ⇌ST cloud	–	SmartThings hub	• ⇌ST cloud
–	Apple HomePod[2]	• ⇌iCloud	–	Philips Hue bridge	• ⇌PH cloud
–	Aqara hub	• ⇌Aqara Cloud; • ⇌HomePod	–	Apple HomePod	• ⇌iCloud
–	Philips Hue bridge	• ⇌PH cloud; • ⇌HomePod	–	Alexa Echo Flex	• ⇌Alexa Cloud
–	Alexa Echo Dot	• ⇌Alexa Cloud	①	ST presence sensor[4]	• ⇌ST hub⇌ST cloud
①	Aqara Mini switch	• ⇌Aqara hub⇌HomePod	②	Kwikset door lock	• ⇌ST hub⇌ST cloud⇌Alexa cloud
②	First Alert smoke sensor	• ⇌ST hub⇌ST cloud; • ⇌ST hub⇌Homebridge⇌HomePod	③	Arlo Essential camera	• ⇌Arlo cloud⇌ST cloud
③④	SmartThings outlet	• ⇌ST hub⇌ST cloud	④	PH motion sensor	• ⇌ST hub⇌ST Cloud
⑤⑥	Wemo smart plug	• ⇌WM cloud⇌Alexa cloud; • ⇌ST hub⇌ST cloud	⑤-⑦	PH motion sensor	• ⇌ST hub⇌ST Cloud; • ⇌ST hub⇌Homebridge⇌HomePod
⑦-⑨	PH motion sensor	• ⇌ST hub⇌ST Cloud; • ⇌ST hub⇌Homebridge⇌HomePod	⑧	PH motion sensor	• ⇌PH bridge⇌HomePod
⑩⑪	ST multipurpose sensor	• ⇌ST hub⇌ST cloud	⑨-⑪	Philips Hue bulb	• ⇌PH bridge⇌ST hub⇌ST Cloud; • ⇌PH bridge⇌PH cloud⇌Alexa Cloud
⑫⑬	Kwikset door lock	• ⇌ST hub⇌ST cloud	⑫	WeMo smart plug	• ⇌HomePod
⑭	VOCOlinc humidifier	• ⇌HomePod	⑬-⑮	WeMo smart plug	• ⇌ST hub⇌ST cloud; • ⇌HomePod
⑮-⑲	Philips Hue bulb	• ⇌PH bridge⇌HomePod; • ⇌PH bridge⇌ST hub⇌ST cloud⇌Alexa cloud	⑮	WeMo smart plug	• ⇌ST hub⇌ST cloud⇌Alexa cloud
⑳	Eve Energy triple outlet[3]	• ⇌HomePod	⑯	VOCOlinc humidifier	• ⇌HomePod
㉑	Garadget door opener	• ⇌Garadget cloud⇌ST cloud; • ⇌Garadget cloud⇌ST cloud⇌ST hub⇌Homebridge⇌HomePod	⑰	WeMo smart plug	• ⇌ST Hub⇌ST cloud
㉒	ST water sensor	• ⇌ST hub⇌ST cloud	⑱	First Alert smoke sensor	• ⇌ST hub⇌ST cloud; • ⇌ST hub⇌Homebridge⇌HomePod
㉓	WeMo smart plug	• ⇌HomePod	⑲	WeMo smart plug	• ⇌HomePod
㉔	WeMo smart plug	• ⇌ST hub⇌ST cloud	⑳	ST motion sensor	• ⇌ST hub⇌ST cloud
㉕	PH motion sensor	• ⇌PH bridge⇌HomePod	㉑	ST multipurpose sensor	• ⇌ST hub⇌ST cloud

[1] • denotes the device itself. For simplicity, router is omitted in the connection path. [2] HomePod is the hub of HomeKit. HomeKit automations run on HomePod, not on iCloud. [3] Connected by a smart heater switch ③, a non-smart microwave and a non-smart oven. [4] Carried with a person.



(a) T_1 (b) T_2

Fig. 11: The floor plans and device placement in the two testbeds, T_1 and T_2. For brevity, personal devices (e.g., smartphones, tablets, laptops) and IoT hubs are not marked in the floor plans.

certain daily activities such as leaving/entering home through the garage door, entering a specific room, etc. In particular, when triggering Rules 14, 15 in T_1 and Rules 8, 9 in T_2, which are triggered by smoke or water leaks, we ask the testbed members to physically trigger the smoke and water sensors in a safe manner. Also, due to the physical restrictions on installing the water valve and sprinkler in the testbeds, we use these rules to control smart switches instead of the real water valve and sprinkler. Although the victim rules are being triggered by the testbed members, our attack script runs to attack the victim rules by delaying the actual device events (e.g., smoke) or commands (see Table IV). To obtain the ground truth for validating if the attacks are successful, we repeat the above process for two days. On the first day, we set the delay period to 0 for all events and commands, i.e., no attacks are conducted. On the second day, we perform DAI attacks by setting some specific time periods to delay the target event or command of the victim rules. We collect the event and command logs for analyzing automation results. The automation result on the first day is used as the *ground truth* for comparison. We compare the automation result of each pair of victim rules on the second day (with attack) with those from the first day (without attack). If the results are the same, the rules are not attacked on the second day. Otherwise, the rules may be attacked. To confirm, we manually check whether the automation result on the second day is consistent with the expected attack result; if so, the attack is validated.

Device Logs. Device logs are needed to evaluate the correctness of automation. Among the nine platforms deployed in the testbeds, three of them (i.e., SmartThings, HomeKit, and Alexa) are used as non-endpoint platforms which have access to a broader range of device types. Alexa does not provide a convenient logging tool. Therefore, we choose to use the built-in logging functions on the SmartThings mobile app and on a third-party mobile app, `Home+ 4`, that can access and export the HomeKit data. Since the Alexa devices in the two testbeds can be accessed by SmartThings and/or HomeKit, all device events in the testbeds can be collected by at least one of the two methods. Note that we do not collect logs of hub devices since they are not used in automation rules. For the convenience of analysis, we convert the raw event logs from the SmartThings and Home+ 4 apps, denoted as $\mathbb{E}_{ST}$, $\mathbb{E}_{HK}$, to a uniform format. Each element (event) in the reformatted logs is a tuple: $\langle$`TestbedID`, `DeviceID`, `Attribute`, `Value`, `Timestamp`$\rangle$, where the combination of `TestbedID`, `DeviceID`, `Attribute` and `Value` uniquely identifies an event type. For example, a *motion active* event sent by device ⑦ in testbed T_1 can be denoted as $\langle 1, ⑦, \text{motion}, \text{active}\rangle$. An event type may have multiple instances at different `Timestamps`. The new event log is sorted by `Timestamp`. Note that the timestamps of events are the time instances when the platforms receive the events, which may have been delayed by an attack. This is the reason why we use events on the first day, in the absence of attacks, as ground truth. Based on the event logs $\mathbb{E}_{ST}$ and $\mathbb{E}_{HK}$ on both days, we can easily track the execution of rules as in existing work [26], [51].

TABLE III: Installed rules in all testbeds. **RID**: rule ID.

Testbed	RID	Rule Description and Device Binding	Platform
T_1	1	When 6pm, if motion ⑦ is active in living room, turn on the ceiling light ⑯.	HomeKit
	2	When 6pm, if no motion ⑦ in living room, toggle ceiling light ⑯ every 15 minutes to simulate occupancy. Press an app button to stop.	SmartThings
	3	When the hall door ⑪ is closed, if the home is in *away* mode, close garage door ㉑, turn on outlets ④⑤⑥ and set to *home* mode.	SmartThings
	4	When the hall door ⑪ is closed, if the home is in *home* mode, open the garage door ㉑.	HomeKit
	5	When the button ① is pressed, set to *away* mode.	HomeKit
	6	When the user (smartphone as presence sensor) leaves, if the home is in *away* mode, close the garage door ㉑.	HomeKit
	7	When the garage door ㉑ is closed, if the home is in *away* mode, lock the front door ⑫ and hall door ⑬.	SmartThings
	8	When kitchen time (12pm, 7pm), if motion ⑨ is active in the kitchen, turn on the heater switch ③.	SmartThings
	9	When power ⑳ exceeds 2500W, if motion ⑦-⑨ is inactive in all rooms, turn off the outlet ⑳.	HomeKit
	10	When the front door ⑩ is opened, if the home is in *away* mode, turn on outlets ④⑤⑥ and set to *home* mode.	SmartThings
	11	When motion ⑦ is detected in living room, turn on the humidifier ⑭, ceiling lamp ⑯ and floor lamp ⑮.	HomeKit
	12	When motion ⑧ is detected in bedroom, if luminance ㉕ is below 15 lux, turn on the ceiling lamp ⑰ and floor lamp ⑱.	HomeKit
	13	When motion ⑧ is detected in bedroom, turn on the ceiling lamp ⑰.	Philips Hue
	14	When smoke ② is detected in kitchen, if the user (smartphone as presence sensor) is off, turn on the sprinkler ㉔.	SmartThings
	15	When water leak ㉒ is detected in kitchen, if no smoke ② is detected, close the water valve ㉓.	HomeKit
	16	When the user (smartphone as presence sensor) leaves, turn off the humidifier ⑭, lights ⑮⑯⑰⑱⑲, and plugs ④⑤⑥.	HomeKit
	17	When 11pm, turn off the humidifier ⑭.	HomeKit
	18	Say "*Alexa, good morning*" to turn on light ⑰.	Alexa
	19	Say "*Alexa, good night*" to turn off the lights ⑮⑯⑰⑱⑲.	Alexa
T_2	1	When user ① arrives, unlock the front door lock ②.	SmartThings
	2	When the door lock ② is unlocked, turn on the surveillance camera ③; when the lock ② is locked, turn off the camera ③.	SmartThings
	3	When luminance ⑤ exceeds 20 lux, if motion ⑧ is inactive, turn off the living room light ⑨.	HomeKit
	4	When luminance ⑤ drops below 20 lux, if the user ① is at home, turn on the living room light ⑨.	SmartThings
	5	When front door ㉑ is opened, if motion ④ is active, turn on outlets ⑬⑭.	SmartThings
	6	When motion ⑦ is detected in study room, turn on the humidifier ⑯.	HomeKit
	7	When user ① leaves, close the water valve ⑰ and lock the door ②.	SmartThings
	8	When smoke ⑱ is detected, open the water valve ⑰.	SmartThings
	9	When smoke ⑱ is detected, open the sprinkler ⑲.	HomeKit
	10	When 6pm, turn on the heater switch ⑫.	HomeKit
	11	When temperature ⑥ exceeds $75^\circ F$, if the user ① is at home, open the window ⑮.	SmartThings
	12	When motion ⑳ is detected in bathroom, turn on the bathroom light ⑩.	SmartThings
	13	When no motion ⑳ is detected in bathroom, turn off the bathroom light ⑩.	SmartThings
	14	When the user leaves (smartphone as presence sensor), turn off the humidifier ⑯ and outlets ⑫⑬⑭.	HomeKit
	15	Say "*Hey Siri, turn off the heater*" to turn off the heater outlet ⑫.	HomeKit
	16	Say "*Hey Siri, turn off the humidifier*" to turn off the humidifier ⑯.	HomeKit
	17	Say "*Alexa, good night*" to turn off the lights ⑨⑩⑪ and close the window ⑮.	Alexa

TABLE IV: A summary of the details for attacking the victim rules in the testbeds, including delayed devices and events/commands, the channels which are utilized to realize the delay, and the delay range that have been tested. COA: Condition Overlapping Attack.

Testbed	Victim Rules[1]	Attack Type	Event/Command to Delay?[2]	Which Channel to Delay?	Delay Range (seconds)
T_1	1 & 2	COA (Action Conflict)	*motion active* event from ⑦	SmartThings hub → router	16-47
	3 & 4	Condition Enabling Attack	*door closed* event from SmartThings	Homebridge → HomePod	10-Unbounded[3]
	6 & 7	Condition Diverging Attack	*away mode* event from HomeKit	Homebridge → SmartThings hub	10
	8 & 9	COA (Chained Execution)	*motion active* event from ⑨	Homebridge → HomePod	10–Unbounded
	10 & 11	Action Disordering Attack	*door open* event from ⑩	SmartThings hub → router	16-47
	12 & 13	Condition Disabling Attack	*motion active* event from ⑧	Philips Hue bridge → HomePod	10–Unbounded
	14 & 15	Trigger-Condition Overlapping Attack	*smoke* event from ②	Homebridge → HomePod	10–Unbounded
T_2	1 & 2	Action Delaying Attack	*turn on* command to ③	router → Arlo camera	120-600
	3 & 4	COA (Infinite Loop)	*motion active* event from ⑧	Philips Hue bridge → HomePod	10–Unbounded
	5 & 6	Action Disordering Attack	*door open* event from ㉑	SmartThings hub → router	16-47
	8 & 9	Action Delaying Attack	*switch on* event from ⑰	SmartThings hub → router	16-47

[1] See Table III for the rules referred by the given RIDs. [2] See Table II for the devices referred by the given device IDs. [3] The upper bound is non-deterministic because it depends on the HomePod's dynamic behavior in runtime.

Delay Range. Although larger event/command delays usually create larger time windows for performing DAI attacks, the allowable delay length (without causing timeouts) is determined by the implementation of the IoT devices and/or platforms. Delaying the communication for too long will usually trigger timeout behaviors from either the device or platform side. We obtain the allowable delay range of device events and commands by reviewing the specification [52] and analyzing the traffic patterns on the target channels shown in Table IV. We find that the allowable delay range is typically determined by three factors: (1) the interval of periodical messages between an IoT device/hub and a platform, e.g., *keep-alive requests* (a.k.a., heartbeat) or *platform-initiated requests*; (2) the maximum allowable delay of *keep-alive reply*; (3) the maximum allowable delay of *event/command reply*.

The SmartThings hub sends a keep-alive request to the SmartThings cloud if it has not sent any keep-alive request or device events to the cloud (i.e., the session is idle) for 31 seconds. Then, it sets a timer for 16 seconds and waits for the keep-alive reply. If the hub does not receive a keep-alive reply when the timer fires, it will trigger a timeout and disconnect the TCP connection with the cloud. On the other hand, the cloud also listens to the session. If the session is idle for 31 seconds, it will also set a 16-second timer. The cloud disconnects the TCP connection with the hub if it does not receive any message from the hub when the timer fires. The actual maximum delay of an event in runtime is dynamic because it also depends on the timing of the event, i.e., the temporal distance between the event and the last message on the session. An attacker needs to release the event (as well as

other messages in the message queue) if he finds that the hub or cloud is about to trigger a timeout. Therefore, the delay range is predictable (between 16-47 seconds) but the attacker needs to dynamically adjust the actual delay length to avoid causing timeout behaviors which may trigger alarms. Philips Hue bridge/Homebridge and HomePod (hosting HomeKit) do not exchange periodic keep-alive messages. Homepod only occasionally initiates a request to query the states of devices that are connected to the bridge and allows for 10-second delay for the reply. According to the HomeKit Accessory Protocol (HAP [52], used by HomeKit), devices, after sending an event message, do not get a reply from HomeKit (on HomePod). Thus, we can delay the events from a Philips Hue bridge or Homebridge until HomeKit requests the device's state (which requires a reply within 10 seconds). Thus, the delay range has a lower bound of 10 seconds and an unbounded upper bound since the HomeKit-initiated request is unpredictable. According to our tests, events can be delayed by more than 10 minutes. In similar ways, we obtain the delay range for other channels. The results are presented in the last column of Table IV.

2) Verifying Results: All attacks listed in Table IV are successfully verified, as shown in Table V. The automation results also show that successful attacks lead to annoyance, inconvenience, and even severe safety threats to the smart home owners. Aside from the verification of the testbeds, we also verify the attack results in a controlled environment, by observing the physical states of devices instead of the IoT events/commands. All the cases in Table IV are physically verified, except for Rules 10 and 11 in testbed T_1 and Rules 5 and 6 in testbed T_2. During the attack, the humidifiers and lights are offline and cannot receive the turn-on command from Rule 11 in T_1 (or Rule 6 in T_2) because their outlets are still in OFF status. When the outlets are turned on, the humidifiers and lights will not receive another turn-on command. As a result, Rule 11 in T_1 (or Rule 6 in T_2) fails to work. The humidifiers were indeed not turned on. Interestingly, we observe that the lights (bulbs) are forced to turn on when the connecting outlets turn on. Thus, the attack does not disable the bulbs from being turned on, but only delays them for several seconds.

C. Attack Opportunities

As discussed in Section IV, some of the DAI attacks are opportunistic and can only be successful when a homeowner behaves in certain manners. To evaluate the possibility of DAI attacks on smart homes, we run the two testbeds on a natural daily basis for one week, without providing any guidelines or restrictions on the daily activities to the testbed members. Infrequent automation rules, i.e., Rules 14 & 15 in T_1 and Rules 8 & 9 in T_2, are excluded in this experiment since they are hardly ever triggered (the triggers are either smoke or water leaks). With the experiment, we aim to answer this question: "What are the opportunities an attacker has to attack the victim rules and what is the success rate of the attack, when the testbed members behave in a natural way?"

1) Methodology: We run both testbeds for one week and collect the device events, without performing any attacks. The collected event logs are transformed to a uniform format $\langle$`TestbedID`, `DeviceID`, `Attribute`, `Value`, `Timestamp`$\rangle$ (same format as in Section V-B). By traversing the event logs, we are able to track the executions of each automation rule. We record each execution instance as a tuple $\langle$`TestbedID`, `RuleID`, `Timestamp`$\rangle$. By doing this, we obtain the collection of execution tuples of every rule. Generally speaking, DAI attacks can only be performed when victim rules are triggered by users. To find out attack opportunities in the week, we perform a case-by-case analysis on the combination of the event log and the execution tuples of every pair of victim rules (listed in Table IV). Consider the victim rules 6 and 7 in testbed T_1. For each execution tuple of Rule 6 in testbed T_1 (i.e., $\langle 1, 6, t_1 \rangle$), we examine if there is an execution tuple of Rule 7 (i.e., $\langle 1, 7, t_2 \rangle$), such that $0 \leq t_2 - t_1 \leq 2$ (in seconds), and if there is a door-closed event of the garage door ㉑ (i.e., $\langle 1, ㉑, \text{door}, \texttt{closed}, t_3 \rangle$) in the event log, such that $t_1 \leq t_3 \leq t_2$. If true, it means that Rule 6 closed the garage door, which in turn triggered the execution of Rule 7, which is an interaction that can be attacked by the DAI attacks; we mark the executions of both rules as an *attack opportunity*.

As discussed in Section IV, to successfully launch an attack, the attacker need to delay specific events by a sufficient period. For Rules 6 and 7, we trace back the event logs to find the most recent *away* mode event (denoted as $\langle 1, \text{N/A}, \text{mode}, \text{away}, t_4 \rangle$). If the time difference between t_2 and t_4 is smaller than a threshold $\Delta T'$ (i.e., the maximum allowable delay of the *away mode* event), i.e., $t_2 - t_4 < \Delta T'$, Rules 6 & 7 can be attacked successfully; otherwise, the attack fails. In this way, we obtain the number of *attack opportunities* and *successful attacks* for every victim rule pair.

Note that, in order to evaluate attack opportunities (i.e., number of successful attacks), we choose not to perform real attacks on the testbeds, for two major reasons. First, real attacks can cause severe safety issues to the testbed members. Second, the attacks against different victim rule pairs may have conflicts in delaying a specific channel.

2) Results: In Table VI, we present the number of attack opportunities and successful attacks over a week. The victim rule pairs are triggered between 3 to 12 times, creating sufficient opportunities for the attacks. The numbers of successful attacks that took place each day are also presented.

The average success rate of all attacks is 0.864. Most attacks have a success rate of 1.00, i.e., they can successfully cause CRI on the victim rules. The attacks on Rules 6 & 7, 8 & 9 in testbed T_1 and Rules 5 & 6 in T_2 fail several times due to the restriction of the allowed delay $\Delta T_{allowed}$ (see Section IV-D about *factors affecting attack successes*). For example, an attack attempts to change the execution order of Rules 5 & 6 in T_2 by delaying the door-open event. Assuming the interval between the time she opens the door and that she enters the study room is $\Delta T_{interval}$, if $\Delta T_{interval} > \Delta T_{allowed}$, the attack fails, because it cannot delay the door-open event

TABLE V: Results of attack validation. See Table II for the definition of device IDs.

Testbed	Victim Rules	Automation Result on Day 1 (without attack)	Automation Result on Day 2 (with attack)	Attack Validated?
T_1	1 & 2	Light ⑯ turns on once.	Light ⑯ turns and off alternately about every 15 minutes.	✓
	3 & 4	Garage door ⑯ is closed.	Garage door ⑯ is closed and then opened.	✓
	6 & 7	Locks ⑫ and ⑬ are locked.	Locks ⑫ and ⑬ are NOT locked.	✓
	8 & 9	Outlet ⑳ remains on when heater ③ turns on.	Outlet ⑳ turns off 2 seconds after heater ③ turns on.	✓
	10 & 11	Outlets ④⑤⑥ and then (humidifier ⑭, lights ⑮⑯) turn on.	Only outlets ④⑤⑥ turn on.	✓
	12 & 13	Both lights ⑰⑱ turn on.	Only ⑰ turns on.	✓
	14 & 15	Sprinkler ㉔ turns on and water valve ㉓ remains opened.	Water valve ㉓ is closed after Sprinkler ㉔ turns on.	✓
T_2	1 & 2	Camera ③ turns on immediately after lock ② is unlocked.	Camera ③ turns on 538 seconds after lock ② is unlocked.	✓
	3 & 4	Light ⑨ turns on.	Light ⑨ turns on and off alternately for 141 seconds.	✓
	5 & 6	Outlets ⑬⑭ turn on and then humidifier ⑯ turns on.	Only outlets ⑬⑭ turn on.	✓
	8 & 9	Water valve ⑰ and Sprinkler ⑲ are turned on within 1 second.	Water valve ⑰ turns on 58 seconds after sprinkler ⑲ turns on.	✓

TABLE VI: Results of attack opportunities and success rates during a week. N_o: the total number of attack opportunities over the week. N_s: the total number of successful attacks over the week. COA: Condition Overlapping Attack.

Testbed	Victim Rules	Attack Type	N_o	N_s	Number of Successful Attacks on Each Day							Success Rate
					Mon.	Tue.	Wed.	Thu.	Fri.	Sat.	Sun.	
T_1	1 & 2	COA (Action Conflict)	3	3	0	1	0	0	0	1	1	1.00
	3 & 4	Condition Enabling Attack	6	6	1	1	1	1	1	0	1	1.00
	6 & 7	Condition Diverging Attack	6	3	0	1	1	0	0	0	1	0.50
	8 & 9	COA (Chained Execution)	7	4	0	1	1	0	0	2	0	0.57
	10 & 11	Action Disordering Attack	2	2	0	0	0	0	1	0	1	1.00
	12 & 13	Condition Disabling Attack	11	11	1	1	2	1	1	3	2	1.00
T_2	1 & 2	Action Delaying Attack	9	9	1	1	1	2	1	1	2	1.00
	3 & 4	COA (Infinite Loop)	12	12	2	2	1	1	1	2	3	1.00
	5 & 6	Action Disordering Attack	7	5	0	1	0	1	1	1	1	0.71

TABLE VII: Results of applying jamming techniques to replicate the attacks in Table IV.

Testbed	Victim Rules	Replicable?	Victim Rules	Replicable?
T_1	1 & 2	✓	3 & 4	✗
	6 & 7	✓	8 & 9	✓
	10 & 11	✗	12 & 13	✓
	14 & 15	✓		
T_2	1 & 2	✗	3 & 4	✓
	5 & 6	✗	8 & 9	✗

(which triggers Rule 5) as much as until the user enters the study room (which triggers Rule 6). Note that DAI attacks are independent in general, unless two concurrent attacks require different delays on the same event(s) or command(s). When an attack fails, no TCP session is disconnected and the attacker can make attempts to perform other attacks seamlessly.

In short, the results in Table VI show that the attacker has sufficient opportunities to launch attacks, raising severe safety and security concerns.

D. Comparison with Jamming Attacks

Jamming does not require the victim home's WiFi password. We thus evaluate whether jamming can be used to construct DAI attacks. Two jamming techniques are investigated. The first one is WiFi micro-jamming [53], which can slightly delay the wireless communications by switching on and off transmitting disruptive signals in high frequency. The test result shows that the introduced delay (10 millisecond or so) is too short to conduct effective attacks. Another technique [54] jams wireless communication by cramming the wireless medium with random frames. Following this technique, we use an *Alfa AWUS036NHA* wireless adapter that is powered by the open-source code [55] to jam WiFi frames. For each attack listed in Table IV, we use jamming to discard (i.e., infinitely delay) the events/commands listed in the **Event/Command to Delay?** column and observe the consequences.

The results in Table VII show that jamming replicates some of the attacks on the victim rule pairs (6 out of 11). This indicates jamming, as a low-level attack method alternative to TCP hijacking and ARP spoofing, can be used to attain the same effect of some DAI attacks, which is alarming. The differences of the two attack methods are as follows. First, the TCP-hijacking based method only delays events/commands but does not discard them, while jamming that discards messages can only construct some of the attack types. For example, if a trigger event is discarded (rather than delayed), the subscribing rule will not be triggered. Second, general jamming frequently causes TCP timeout and disconnection alerts. We discuss reactive jamming that does not discard messages or cause disconnection in Section VI-B.

VI. Discussion

A. Countermeasures Against DAI Attacks

First, the smart home end users can raise the bar for attackers to intrude into the IoT network by using strong WiFi passwords and setting up an isolated sub-network for IoT if they share a WiFi network with other people. Also, vendors of IoT devices/hubs and WiFi routers should enhance the security of their products. For example, we find that many IoT devices and home routers are not resistant to ARP spoofing (also verified in the IoT Inspector project [50]), although effective defenses against ARP spoofing exist. One possible reason that explains the wide feasibility of ARP spoofing is that TLS gives the illusion of "sufficient" protection under traffic hijacking

attacks, while our work illustrates the contrary. We estimate there is a long way to go to eliminate ARP spoofing attacks.

Second, device vendors and platform providers can reduce the intervals of TLS-protected heartbeat (a.k.a., keep-alive) messages and enforce two-way liveness checking of event and command messages, which can significantly reduce the allowed delays. However, this requires IoT vendors to modify their protocols and device firmware. Plus, more frequent heartbeat messages lead to a higher overhead, which should be considered carefully in the IoT design.

Third, there are known synchronization and recovery techniques for handling inconsistency and disorder issues in distributed systems [56], [57], [58]. A challenge to deploy such techniques in IoT is the lack of a central entity that has a global view and control on heterogeneous IoT devices and multiple proprietary IoT platforms, or a mechanism for distributed platforms to collaboratively synchronize correct observations on devices in the fragmented IoT ecosystem.

Fourth, researchers can design approaches to detecting potential DAI attacks. Various techniques in the state-of-the-art works [6], [47], [26], [5] can be utilized to extract deployment information (such as devices, rules, and platforms) from a given smart home system. After incorporating the extracted information into our formal model, the *observation equivalence* based technique (described in Section IV-A and Appendix C) can be used to detect potential DAI attacks by verifying whether any pair of rules are not *CRI-resistant* in the presence of delay attacks. The detection result can be presented to the user, who can then re-configure the rules. An alternative mitigation strategy is to enforce security properties/goals, which prevent devices from transitioning into unsafe states [8].

B. Other Approaches to Introduce Delay

TCP hijacking and ARP spoofing are not the only way to introduce delays. A more sophisticated jamming, *reactive jamming*, is another promising approach. A reactive jammer can recognize events or commands from encrypted IEEE 802.11 traffic [29] and jam selective events/commands in a smart manner, although it requires specific hardware; that is, by exploiting the retransmission mechanism of TCP protocol, the jammer could block the first $N-1$ retransmissions of a target event/command and let the N-th retransmission pass, such that TCP timeout is not triggered due to the failed first $N-1$ retransmissions but will be triggered if the N-th retransmission also fails. Thus, a delay equal to the elapsed time from the first to the N-th retransmission is injected to the transmission process. Different from ARP spoofing, which can be launched from an ordinary WiFi device, reactive jamming needs dedicated hardware with high sensitivity and computation capability to recognize and jam the target event/command before it is delivered to the receiver. We leave the reactive-jamming based implementation as the future work.

VII. Related Work

A. Synchronization in IoT

Synchronization is critical in smart home IoT systems since IoT devices, platforms and mobile apps interact closely with each other in smart homes. Zhou et al. [59] identify that the working state transitions of devices, mobile apps, and clouds are not properly safeguarded. By triggering and exploiting the out-of-synchronization bugs, attackers can remotely harm the system, including taking over devices and replacing them with fake ones. This paper studies a different topic: delaying the transmission of IoT events/commands and the exploitation. OConnor et al. [60] utilize the design flaws in the telemetry of IoT devices to block the delivery of sensor measurements to IoT servers or commands to actuators, causing synchronization problems between devices and clouds. In contrast to the jamming or discarding-message based attacks [60], our work exploits delays, and does not raise alarms at any layers of the IoT protocol stack or rely on implementation bugs.

B. IoT Security and Privacy

IoT Security and privacy have been studied in various aspects, such as platforms, apps, devices, and data. Fernandes et al. [61] and Mi et al. [62] unveil the vulnerabilities on prominent IoT platforms, SmartThings and IFTTT, respectively. The IoT app-level security is intensively studied by recent work [63], [38], [39], [64], [65]. Regarding IoT devices, solutions are proposed to enhance the authentication [66], [67], access control [68], [69], etc. Researchers also employ data-driven techniques to detect device anomalies [70], [71], [72]. Fu et al. [72] design HAWatcher, which utilizes rich semantic information to mine correlations in smart environments, and achieves highly-accurate and explainable anomaly detection. A number of approaches [73], [51], [74], [75] are designed to protect privacy-sensitive IoT data. Chi et al. [51] propose PFirewall, which enforces data-minimization policies to significantly reduce the disclosure of IoT data and protect users' privacy from IoT platforms, without changing IoT devices or platforms. Despite the vast amount of work, little research has been done to uncover the unique threats in multi-platform systems. Yuan et al. [76] reported the flaws in IoT device access delegation across multiple IoT clouds. We are the first to study unique delay-derived security threats in multi-platform systems.

C. Cross-Rule Interference Problems

Cross-Rule Interference (CRI) problems have attracted much attention of IoT security researchers. A lot of works are engaged to categorize [5], [9], understand [40] and detect [5], [6], [7], [9], [77], [8], [10] the CRI problems. Our prior work [4], [5] is the earliest one that comprehensively categorizes and formally describes CRI threats. In these works, CRI problems are caused by users who misconfigure the automation rules for their own smart homes. However, all these works only consider CRI in single-platform systems, implicitly assuming consistent and order-preserving observations on the device states. Although IoTGuard [8] and IoTIE [11] configure

rules on two platforms, SmartThings and IFTTT, for evaluation, they do not take multi-platform or its unique features into consideration, while analyzing the CRI problems. Moreover, both works convert IFTTT rules into equivalent SmartThings apps, and use SmartThings to run all the rules; essentially, they still make the same assumptions. Our work is the first that studies CRI problems in more complex but realistic system models, where multiple event/command transmission paths and/or multiple platforms coexist. Our work is also the first that introduces the *delay* factor into the investigation of the CRI problem, and it reveals and demonstrates a family of new attacks that are ignored by existing CRI detection solutions.

VIII. Responsible Disclosure

We have reported the event/command delay attacks and possible exploits to IoT vendors: Google, SimpliSafe, Apple (HomeKit), Ring and SmartThings. Google, SimpliSafe and Ring acknowledged the problem. Google and SimpliSafe expressed they would conduct a bug/vulnerability fixing procedure with the product team. Ring said they had planned a mitigation that make side channel attacks more difficult. Apple regards the reported attacks as "*expected behavior and working as designed*", although Apple's HomeKit allows the longest delay window (tens of minutes or even longer).

IX. Conclusion

We studied unique cross-rule interference (CRI) problem due to the inconsistency and disorder issues in single-platform multi-path and multi-platform smart home systems. We uncovered that selective messages delay attacks have detrimental impacts on smart home automation, causing various CRI problems that cannot be detected by existing detectors. We revealed seven categories of such attacks, referred to as Delay-based Automation Interference (DAI) attacks, which were analyzed and demonstrated using two smart homes. The evaluation results show that DAI attacks of all the seven categories can be launched, and an attacker can conduct DAI attacks with a high success rate, without raising alerts in any layer of the current IoT protocol stack.

Acknowledgement

This work was supported in part by the US National Science Foundation (NSF) under grants CNS-1828363, CNS-2204785, CNS-2205868, CNS-1856380, CNS-2016415, and CNS-2107093.

References

[1] "Apple HomeKit," https://www.apple.com/ios/home/, 2020.
[2] "SmartThings," https://www.smartthings.com/, 2020.
[3] "Amazon Alexa," https://developer.amazon.com/en-US/alexa/devices/smart-home-devices, 2020.
[4] H. Chi, Q. Zeng, X. Du, and J. Yu, "Cross-app interference threats in smart homes: Categorization, detection and handling," *arXiv preprint arXiv:1808.02125*, 2018.
[5] ——, "Cross-app interference threats in smart homes: Categorization, detection and handling," in *50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2020.
[6] Z. B. Celik, P. McDaniel, and G. Tan, "Soteria: Automated iot safety and security analysis," in *Usenix Security Symposium*, 2018.
[7] D. T. Nguyen, C. Song, Z. Qian, S. V. Krishnamurthy, E. J. Colbert, and P. McDaniel, "Iotsan: fortifying the safety of iot systems," in *ACM CoNEXT*, 2018.
[8] Z. B. Celik, G. Tan, and P. McDaniel, "IoTGuard: Dynamic enforcement of security and safety policy in commodity iot," in *NDSS*, 2019.
[9] Q. Wang, P. Datta, W. Yang, S. Liu, A. Bates, and C. A. Gunter, "Charting the attack surface of trigger-action iot platforms," in *ACM CCS*, 2019.
[10] K.-H. Hsu, Y.-H. Chiang, and H.-C. Hsiao, "Safechain: Securing trigger-action programming from attack chains," *IEEE Transactions on Information Forensics and Security*, 2019.
[11] Z. Chen, F. Zeng, T. Lu, and W. Shu, "Multi-platform application interaction extraction for iot devices," in *IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, 2019.
[12] M. Balliu, M. Merro, and M. Pasqua, "Securing cross-app interactions in iot platforms," in *IEEE CSF*, 2019.
[13] M. Alhanahnah, C. Stevens, and H. Bagheri, "Scalable analysis of interaction threats in iot systems," in *ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2020.
[14] "A comprehensive guide to smart home device compatibility," https://www.adt.com/resources/smart-home-device-compatibility, 2021.
[15] "Fragmentation in IoT – one roadblock in IoT deployment," https://www.cleantech.com/fragmentation-in-iot-one-roadblock-in-iot-deployment/, 2017.
[16] C. Fu, Q. Zeng, H. Chi, X. Du, and S. L. Valluru, "Iot phantom-delay attacks: Demystifying and exploiting iot timeout behaviors," in *Technical Report*, 2021.
[17] "AWS IoT," https://docs.aws.amazon.com/iot/latest/developerguide/what-is-aws-iot.html, 2021.
[18] L. Yu, B. Luo, J. Ma, Z. Zhou, and Q. Liu, "You are what you broadcast: Identification of mobile and iot devices from (public) wifi," in *USENIX Security Symposium*, 2020.
[19] M. Miettinen, S. Marchal, I. Hafeez, N. Asokan, A.-R. Sadeghi, and S. Tarkoma, "Iot sentinel: Automated device-type identification for security enforcement in iot," in *IEEE ICDCS*, 2017.
[20] B. Bezawada, M. Bachani, J. Peterson, H. Shirazi, I. Ray, and I. Ray, "Iotsense: Behavioral fingerprinting of iot devices," *arXiv preprint arXiv:1804.03852*, 2018.
[21] A. K. Dalai and S. K. Jena, "Wdtf: A technique for wireless device type fingerprinting," *Wireless Personal Communications*, vol. 97, no. 2, pp. 1911–1928, 2017.
[22] Y. Meidan, M. Bohadana, A. Shabtai, M. Ochoa, N. O. Tippenhauer, J. D. Guarnizo, and Y. Elovici, "Detection of unauthorized iot devices using machine learning techniques," *arXiv preprint arXiv:1709.04647*, 2017.
[23] M. R. Shahid, G. Blanc, Z. Zhang, and H. Debar, "Iot devices recognition through network traffic analysis," in *IEEE International Conference on Big Data (Big Data)*, 2018.
[24] K. Yang, Q. Li, and L. Sun, "Towards automatic fingerprinting of iot devices in the cyberspace," *Computer Networks*, vol. 148, pp. 318–327, 2019.
[25] N. Apthorpe, D. Reisman, S. Sundaresan, A. Narayanan, and N. Feamster, "Spying on the smart home: Privacy attacks and defenses on encrypted iot traffic," *arXiv preprint arXiv:1708.05044*, 2017.
[26] W. Zhang, Y. Meng, Y. Liu, X. Zhang, Y. Zhang, and H. Zhu, "Homonit: Monitoring smart home apps from encrypted traffic," in *ACM CCS*, 2018.
[27] 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 Conference on Security and Privacy in Wireless and Mobile Networks*, 2020.
[28] 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.
[29] R. Trimananda, J. Varmarken, A. Markopoulou, and B. Demsky, "Packet-level signatures for smart home devices," *NDSS*, 2020.
[30] T. Gu, Z. Fang, A. Abhishek, and P. Mohapatra, "Iotspy: Uncovering human privacy leakage in iot networks via mining wireless context," in *IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, 2020.
[31] A. Subahi and G. Theodorakopoulos, "Detecting iot user behavior and sensitive information in encrypted iot-app traffic," *Sensors*, vol. 19, no. 21, p. 4777, 2019.

[32] 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*, 2020.
[33] "ARP spoofing," https://www.veracode.com/security/arp-spoofing, 2021.
[34] S. Whalen, "An introduction to arp spoofing," *Node99 [Online Document], April*, 2001.
[35] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis *et al.*, "Understanding the mirai botnet," in *USENIX Security Symposium*, 2017.
[36] "The Mirai IoT botnet holds strong in 2020," https://searchsecurity.techtarget.com/feature/The-Mirai-IoT-botnet-holds-strong-in-2020, 2020.
[37] "The Transport Layer Security (TLS) Protocol Version 1.3," https://tools.ietf.org/html/rfc8446, 2018.
[38] Y. J. Jia, Q. A. Chen, S. Wang, A. Rahmati, E. Fernandes, Z. M. Mao, and A. Prakash, "Contexiot: Towards providing contextual integrity to appified iot platforms," in *NDSS*, 2017.
[39] Q. Wang, W. U. Hassan, A. Bates, and C. Gunter, "Fear and logging in the internet of things," in *Proceedings of The Network and Distributed System Security Symposium*, 2018.
[40] W. Ding and H. Hu, "On the safety of iot device physical interaction control," in *ACM CCS*, 2018.
[41] R. Lanotte and M. Merro, "A calculus of cyber-physical systems," in *Springer International Conference on Language and Automata Theory and Applications*, 2017.
[42] M. Hennessy and T. Regan, "A process algebra for timed systems," *Information and computation*, vol. 117, no. 2, pp. 221–239, 1995.
[43] "ADT security systems," https://www.adt.com/, 2021.
[44] M. Surbatovich, J. Aljuraidan, L. Bauer, A. Das, and L. Jia, "Some recipes can do more than spoil your appetite: Analyzing the security and privacy risks of ifttt recipes," in *International Conference on World Wide Web*, 2017.
[45] M. Palekar, E. Fernandes, and F. Roesner, "Analysis of the susceptibility of smart home programming interfaces to end user error," in *IEEE Security and Privacy Workshops*, 2019.
[46] "Atomi smart tower heater," https://atomismart.com/product/smart-black-tower-heater/, 2021.
[47] W. Ding, H. Hu, and L. Cheng, "IoTSafe: Enforcing safety and security policy with real iot physical interaction discovery," in *NDSS*, 2021.
[48] "Smartthings public github repository," https://github.com/SmartThingsCommunity/SmartThingsPublic, 2020.
[49] "IoTBench test suite," https://github.com/IoTBench/IoTBench-test-suite, 2019.
[50] "IoT Inspector," https://iotinspector.org/, 2021.
[51] H. Chi, Q. Zeng, X. Du, and L. Luo, "PFirewall: Semantics-aware customizable data flow control for smart home privacy protection," in *NDSS*, 2021.
[52] "Homekit accessory protocol," https://developer.apple.com/support/homekit-accessory-protocol/, 2020.
[53] R. Ogen, K. Zvi, O. Shwartz, and Y. Oren, "Sensorless, permissionless information exfiltration with wi-fi micro-jamming," in *12th USENIX Workshop on Offensive Technologies (WOOT)*, 2018.
[54] M. Vanhoef and F. Piessens, "Advanced wi-fi attacks using commodity hardware," in *ACSAC*, 2014.
[55] "Advanced wi-fi attacks using commodity hardware," https://github.com/vanhoefm/modwifi, 2020.
[56] D. P. Reed and R. K. Kanodia, "Synchronization with eventcounts and sequencers," *Communications of the ACM*, vol. 22, no. 2, pp. 115–123, 1979.
[57] R. Strom and S. Yemini, "Optimistic recovery in distributed systems," *ACM Transactions on Computer Systems (TOCS)*, vol. 3, no. 3, pp. 204–226, 1985.
[58] O. Simeone, U. Spagnolini, Y. Bar-Ness, and S. H. Strogatz, "Distributed synchronization in wireless networks," *IEEE Signal Processing Magazine*, vol. 25, no. 5, pp. 81–97, 2008.
[59] W. Zhou, Y. Jia, Y. Yao, L. Zhu, L. Guan, Y. Mao, P. Liu, and Y. Zhang, "Discovering and understanding the security hazards in the interactions between iot devices, mobile apps, and clouds on smart home platforms," in *USENIX Security Symposium*, 2019.
[60] T. OConnor, W. Enck, and B. Reaves, "Blinded and confused: uncovering systemic flaws in device telemetry for smart-home internet of things," in *ACM WiSec*, 2019.
[61] E. Fernandes, J. Jung, and A. Prakash, "Security analysis of emerging smart home applications," in *IEEE Symposium on Security and Privacy (SP)*, 2016.
[62] X. Mi, F. Qian, Y. Zhang, and X. Wang, "An empirical characterization of ifttt: ecosystem, usage, and performance," in *Internet Measurement Conference*, 2017.
[63] L. Luo, Q. Zeng, B. Yang, F. Zuo, and J. Wang, "Westworld: Fuzzing-assisted remote dynamic symbolic execution of smart apps on iot cloud platforms," in *Annual Computer Security Applications Conference (ACSAC)*, 2021.
[64] I. Bastys, M. Balliu, and A. Sabelfeld, "If this then what?: Controlling flows in iot apps," in *ACM CCS*, 2018.
[65] Z. B. Celik, L. Babun, A. K. Sikder, H. Aksu, G. Tan, P. McDaniel, and A. S. Uluagac, "Sensitive information tracking in commodity iot," in *USENIX Security Symposium*, 2018.
[66] X. Li, F. Yan, F. Zuo, Q. Zeng, and L. Luo, "Touch well before use: Intuitive and secure authentication for iot devices," in *ACM MobiCom*, 2019.
[67] X. Li, Q. Zeng, L. Luo, and T. Luo, "T2pair: Secure and usable pairing for heterogeneous iot devices," in *ACM CCS*, 2020.
[68] S. Lee, J. Choi, J. Kim, B. Cho, S. Lee, H. Kim, and J. Kim, "Fact: Functionality-centric access control system for iot programming frameworks," in *Proceedings of the 22nd ACM on Symposium on Access Control Models and Technologies*. ACM, 2017, pp. 43–54.
[69] S. Demetriou, N. Zhang, Y. Lee, X. Wang, C. A. Gunter, X. Zhou, and M. Grace, "Hanguard: Sdn-driven protection of smart home wifi devices from malicious mobile apps," in *ACM Conference on Security and Privacy in Wireless and Mobile Networks*, 2017.
[70] J. Choi, H. Jeoung, J. Kim, Y. Ko, W. Jung, H. Kim, and J. Kim, "Detecting and identifying faulty iot devices in smart home with context extraction," in *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2018.
[71] S. Birnbach, S. Eberz, and I. Martinovic, "Peeves: Physical event verification in smart homes," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2019.
[72] C. Fu, Q. Zeng, and X. Du, "HAWatcher: Semantics-aware anomaly detection for appified smart homes," in *USENIX Security Symposium*, 2021.
[73] R. Xu, Q. Zeng, L. Zhu, H. Chi, X. Du, and M. Guizani, "Privacy leakage in smart homes and its mitigation: Ifttt as a case study," *IEEE Access*, vol. 7, pp. 63 457–63 471, 2019.
[74] X. Liu, Q. Zeng, X. Du, S. L. Valluru, C. Fu, X. Fu, and B. Luo, "Sniffmislead: Non-intrusive privacy protection against wireless packet sniffers in smart homes," in *24th International Symposium on Research in Attacks, Intrusions and Defenses*, 2021, pp. 33–47.
[75] E. Fernandes, J. Paupore, A. Rahmati, D. Simionato, M. Conti, and A. Prakash, "Flowfence: Practical data protection for emerging iot application frameworks." in *USENIX Security Symposium*, 2016, pp. 531–548.
[76] B. Yuan, Y. Jia, L. Xing, D. Zhao, X. Wang, D. Zou, H. Jin, and Y. Zhang, "Shattered chain of trust: Understanding security risks in cross-cloud iot access delegation," in *USENIX Security Symposium*, 2020.
[77] J. L. Newcomb, S. Chandra, J.-B. Jeannin, C. Schlesinger, and M. Sridharan, "IoTA: a calculus for internet of things automation," in *ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, 2017.
[78] R. Focardi and F. Martinelli, "A uniform approach for the definition of security properties," in *Springer International Symposium on Formal Methods*, 1999.
[79] R. Milner, *Communicating and mobile systems: the pi calculus*. Cambridge university press, 1999.

APPENDIX

A. Online Study on Statistics of Smart Home Deployments

We conduct an online user study on Amazon Mechanical Turk to investigate the pervasiveness of different smart home deployment categories in the wild, including single-platform single-path (SPSP), single-platform multi-path (SPMP) and multi-platform (MP) ones. In this study, we follow the acceptable use policy of Amazon Mechanical Turk.

Methodology. We design a survey asking people who have experience with IoT devices and automation to provide information of their own deployments. Specifically, we ask them to answer simple questions such as the number of devices and mobile/web companion apps in their home, and more complex questions by writing down the list of their devices, companion apps, and automation rules. The survey questions and responses are made publicly available at

https://github.com/HaotianChi/SH-survey.git

Results. We receive 85 responses from the survey participants. Through a qualitative analysis on the collected information of devices, companion apps and automation rules, we rebuild the deployment of each participant's home. The result shows that the number of SPSP, SPMP and MP deployments are 15 (17.6%), 17 (20.0%) and 53 (62.4%), respectively. Among the 53 MP deployments, 36 use two platforms, 11 use three, 5 use four and 1 uses five. A total of 82.4% deployments in the survey are SPMP or MP deployments. The results show that the testbed setup in Section V is realistic.

B. Formal Modeling of Smart Home Systems

We use the process calculus [41] to formally model multi-path multi-platform smart home systems and delay-based attacks. The process calculus extends the *timed process calculus* [42] with two action prefixes: reading sensor values and writing values to actuators.

Physical Environment. Let $\hat{\mathcal{F}} \subset \mathcal{F}$ be a set of physical features (e.g., temperature), $\hat{\mathcal{S}} \subset \mathcal{S}$ a set of sensors, $\hat{\mathcal{A}} \subset \mathcal{A}$ a set of actuators. Note that in this paper, each actuatable device (e.g., a lock) is conceptually split into an actuator that executes commands and a sensor that measures the device state. For a set of names $\mathcal{N}$, we define $\mathbb{R}^{\mathcal{N}}$ as the set of functions that assign a real value (broadly, integers and Booleans are replaceable with reals) to each name in $\mathcal{N}$. A physical environment E is a 5-tuple $\langle \xi_f, \xi a, evol, \xi s, meas \rangle$:

- $\xi_f \in \mathbb{R}^{\hat{\mathcal{F}} \cup \hat{\mathcal{A}}}$ is the state (i.e., physical features or actuator values) function that returns the value of the state;
- $\xi_a \in \mathbb{R}^{\hat{\mathcal{A}}}$ is the actuator function that returns the current value of an actuator;
- $evol : \mathbb{R}^{\hat{\mathcal{F}} \cup \hat{\mathcal{A}}} \times \mathbb{R}^{\hat{\mathcal{A}}} \to \mathbb{R}^{\hat{\mathcal{F}} \cup \hat{\mathcal{A}}}$ is the evolution function, which returns the next state function upon changes of actuators;
- $meas : \mathbb{R}^{\hat{\mathcal{F}} \cup \hat{\mathcal{A}}} \to \mathbb{R}^{\hat{\mathcal{S}}}$ is the measurement function that returns the reading of a sensor.

In addition to physical features, we include actuators in the state function to model the dependence between actuators. For example, a smart microwave, even turned on, cannot function if its power cable is connected to a smart plug that is OFF. It is not the focus of this paper to precisely model the physical evolution laws. For the simplicity of analysis, we assume that the evolution function $evol$ only takes as input the state function ξ_f and actuator function ξ_a to determine a new state function.[4] Also, we assume the sensors $\mathcal{S}$ measures physical features (i.e., the $meas$ function) without errors. Local hub devices are omitted since they only forward messages.

[4]In reality, a physical feature (i.e., temperature) may be influenced by other factors (e.g., outdoor temperature, humidity) and has uncertainty.

IoT Processes. Processes are defined in the following syntax:

$$
\begin{aligned}
P, Q \;&::=\; \texttt{nil} \mid \texttt{idle}.P \mid \pi.P \mid P\backslash c \mid P \parallel Q \mid \\
&\quad \pi_1.P + \pi_2.Q \mid \texttt{if}\ (b)\ \{P\}\ \texttt{else}\ \{Q\} \mid H\langle \tilde{w} \rangle \\
\pi, \pi_1, \pi_2 \;&::=\; \texttt{snd}\,\bar{c}\langle v \rangle \mid \texttt{rcv}\,c(x) \mid \texttt{read}\,s(x) \mid \\
&\quad \texttt{read}\,a(x) \mid \texttt{wrt}\,\bar{a}(v) \mid \tau
\end{aligned}
$$

Terminated process is written as nil. The process $\texttt{idle}.P$ continues as P after sleeping for one time unit. Action prefixes (ranged over π, π_1, and π_2) contain six actions: sending values to channels ($\texttt{snd}\,\bar{c}\langle v \rangle$), receiving values from channels ($\texttt{rcv}\,c(x)$), reading new sensor measurements ($\texttt{read}\,s(x)$), reading new actuator states ($\texttt{read}\,a(x)$), and writing values on actuators ($\texttt{wrt}\,\bar{a}(v)$) and unobservable action (τ). $P\backslash c$ behaves as P but the channel name c is restricted to P and can only be used for communications between components within P. We use $P\{v/x\}$ to denote the substitution of the variable x with the value v in P. The process $\pi_1.P + \pi_2.Q$ is called *summations* (or *sums*) and can enact either $\pi_1.P$ or $\pi_2.Q$. The process $P \parallel Q$ denotes the parallel composition of concurrent processes P and Q. The process $\texttt{if}\ (b)\ \{P\}\ \texttt{else}\ \{Q\}$ is a standard conditional statement. $H\langle w_1, \cdots, w_k \rangle$ denotes a recursive process and can be considered as an invocation, with actual parameters $w_1, \cdots, w_k$, of the *process definition* $H(x_1, \cdots, x_k) = P$.

Each platform L has access to a subset of all sensors $\tilde{\mathcal{S}} \subset \hat{\mathcal{S}}$ and actuators $\tilde{\mathcal{A}} \subset \hat{\mathcal{A}}$. The database on the platform maintains a function $\xi'_s \in \mathbb{R}^{\tilde{\mathcal{S}}}$ that returns the current state of each sensor. For convenience, we define two database primitives: a function $value\_in\_db$ that returns the values of specified devices $\tilde{x}$ and a meta-process update_db that updates the value of a sensor:

- $value\_in\_db(\tilde{x}) = \{\xi'_s(x_i) : x_i \in \tilde{x}\}$;
- $\texttt{update\_db}(x, v) \to \xi'_s(x) = v$.

Example. We use an automation rule (*when the homeowner arrives, unlock the door*) on platform A and another one (*when the user says "I am home", if the door is unlocked, lock it*) on platform B as a running example to concretize a system. A presence sensor PS and a smart lock LK connect with a platform A that runs R1, and a smart speaker SP and the smart lock LK connect with platform B running R2. For the sake of brevity, we omit all components (e.g., IoT hubs, routers, endpoint clouds) that only forward messages. Thus, the smart home is $\texttt{Sys} = E \bowtie P$ where the physical environment $E = \langle \xi_f, \xi a, evol, \xi s, meas \rangle$ is defined as

- $\xi_f \in \mathbb{R}^{\{presence, voice, \texttt{LK}\}}$, $\xi_f^0(presence) = \texttt{FALSE}$, $\xi_f^0(voice) = \texttt{EMPTY}$, $\xi_f^0(\texttt{LK}) = \texttt{LOCKED}$
- $\xi_a \in \mathbb{R}^{\{\texttt{LK}\}}$, $\xi_a^0(\texttt{LK}) = \texttt{LOCKED}$
- $evol(\xi_f^i, \xi_a^i) = \{\xi : \xi(\texttt{LK}) = \xi_a^i(\texttt{LK})\}$
- $meas(\xi_f^i) = \{\xi : \xi(\texttt{LK}) = \xi_f^i(\texttt{LK}), \xi(\texttt{PS}) = (\texttt{if}\ \xi_f^i(presence) = \texttt{TRUE DETECTED else CLEAR}), \xi(\texttt{SP}) = (\texttt{if}\ \xi_f^i(\texttt{voice}) = \text{"I am home"}\ \texttt{DETECTED else CLEAR})\}$. The timestamp $i = 0 \cdots n$ denotes the discrete time clock

and the process P is the parallel composition of the following sub-processes;

- $\texttt{Sen}_{\texttt{PS}} = \texttt{read}\,\texttt{PS}(val).\texttt{snd}\,\overline{event}_{\texttt{A}}(\texttt{PS}, val).\texttt{idle}.\texttt{Sen}_{\texttt{PS}}$
- $\texttt{Sen}_{\texttt{SP}} = \texttt{read}\,\texttt{SP}(val).\texttt{snd}\,\overline{event}_{\texttt{B}}(\texttt{SP}, val).\texttt{idle}.\texttt{Sen}_{\texttt{SP}}$
- $\texttt{Sen}_{\texttt{LK}} = \texttt{read}\,\texttt{LK}(val).\texttt{snd}\,\overline{event}_{\texttt{A}}(\texttt{LK}, val).\texttt{snd}\,\overline{event}_{\texttt{B}}(\texttt{LK}, val)$ $.\texttt{idle}.\texttt{Sen}_{\texttt{LK}}$
- $\texttt{Act}_{\texttt{LK}} = \texttt{rcv}\,cmd_{\texttt{A}}(\texttt{LK}, val).\texttt{wrt}\,\overline{\texttt{LK}}(val).\texttt{idle}.\texttt{Act}_{\texttt{LK}}$ $+\ \texttt{rcv}\,cmd_{\texttt{B}}(\texttt{LK}, val).\texttt{wrt}\,\overline{\texttt{LK}}(val).\texttt{idle}.\texttt{Act}_{\texttt{LK}}$
- $\texttt{SrvPlt}_{\texttt{A}} = \texttt{rcv}\,event_{\texttt{A}}(id, val).\texttt{snd}\,\overline{update}_{\texttt{A}}(id, val)$ $.\texttt{snd}\,\overline{trigger}_{\texttt{R1}}(id, val).\texttt{idle}.\texttt{SrvPlt}_{\texttt{A}}$ $+\ \texttt{rcv}\,action_{\texttt{R1}}(id, val).\texttt{snd}\,\overline{cmd}_{\texttt{A}}(id, val).\texttt{idle}.\texttt{SrvPlt}_{\texttt{A}}$
- $\texttt{SrvPlt}_{\texttt{B}} = \texttt{rcv}\,event_{\texttt{B}}(id, val).\texttt{snd}\,\overline{update}_{\texttt{B}}(id, val)$ $.\texttt{snd}\,\overline{trigger}_{\texttt{R2}}(id, val).\texttt{idle}.\texttt{SrvPlt}_{\texttt{B}}$ $+\ \texttt{rcv}\,action_{\texttt{R2}}(id, val).\texttt{snd}\,\overline{cmd}_{\texttt{B}}(id, val).\texttt{idle}.\texttt{SrvPlt}_{\texttt{B}}$
- $\texttt{Rule}_{\texttt{R1}} = \texttt{rcv}\,trigger_{\texttt{R1}}(id, val).\texttt{if}\ (id, val = \texttt{PS}, \texttt{DETECTED})$ $\{\texttt{snd}\,action_{\texttt{R1}}(\texttt{LK}, \texttt{UNLOCKED}).\texttt{idle}.\texttt{Rule}_{\texttt{R1}}\}\ \texttt{else}$ $\{\texttt{idle}.\texttt{Rule}_{\texttt{R1}}\}$
- $\texttt{Rule}_{\texttt{R2}} = \texttt{rcv}\,trigger_{\texttt{R2}}(id, val).\texttt{if}\ (id, val = \texttt{SP}, \texttt{DETECTED})\ \{$ $\texttt{snd}\,\overline{condition}_{\texttt{R2}}(\texttt{LK}).\tau.\texttt{rcv}\,res_{\texttt{R2}}(value\_in\_db(\texttt{LK}))).$ $\texttt{if}\ (value\_in\_db(\texttt{LK}) = \texttt{UNLOCKED})\ \{\texttt{snd}\,\overline{action}_{\texttt{R2}}(\texttt{LK}, \texttt{LOCKED})$ $\}\ \texttt{else}\ \{\texttt{idle}.\texttt{Rule}_{\texttt{R2}}\}\ \}\quad \texttt{else}\ \{\texttt{idle}.\texttt{Rule}_{\texttt{R2}}\}$
- $\texttt{DB}_{\texttt{A}} = \texttt{rcv}\,update_{\texttt{A}}(id, val).update\_db(id, val).\texttt{idle}.\texttt{DB}_{\texttt{A}}$
- $\texttt{DB}_{\texttt{B}} = \texttt{rcv}\,update_{\texttt{B}}(id, val).update\_db(id, val).\texttt{idle}.\texttt{DB}_{\texttt{B}}$ $+\ \texttt{rcv}\,condition_{\texttt{R2}}(id).\tau.\texttt{snd}\,\overline{res}_{\texttt{R2}}(value\_in\_db(id)).\texttt{idle}.\texttt{DB}$

Selective Event/Command Delaying Attack. As discussed in Section II-D, the attacker can hijack and selectively delay specific events or commands on a channel. To model the attack, we write $\texttt{Delay}\,(c, c', id, val, t) = \texttt{rcv}\,c(x, y).\texttt{idle}^{t}.\texttt{snd}\,\bar{c}'\langle x, y\rangle.idle.\texttt{Delay}$ to denote a delay attack process that delays the value-passing on channel c by $t \in \mathbb{N}$ time units. $\texttt{idle}^{t}.P$ is a shorthand for $\texttt{idle}.\cdots.\texttt{idle}.P$ where $\texttt{idle}$ appears t consecutive times. t is a positive integer if the received data (x, y) from c is an event (command) which is produced by (destined to) a device with identifier id and has a value equal to val; otherwise, t is equal to 0. Thus, a smart home system $\texttt{Sys} = E \bowtie P$ being attacked becomes $\texttt{Sys}^{(\tilde{t})} = E \bowtie P^{(\tilde{t})}$, where $P^{(\tilde{t})} = P(\tilde{c} \to \tilde{c}') \parallel \texttt{Delay}\,(\tilde{c}, \tilde{c}', \tilde{id}, \tilde{val}, \tilde{t})$. Specifically, $P(\tilde{c} \to \tilde{c}')$ substitutes new channels $\tilde{c}'$ for ones $\tilde{c}$ that are used by the $\texttt{rcv}$ actions in P. The sub-processes that read from the attacked channels $\tilde{c}$ are converted to ones that read from the corresponding channels $\tilde{c}'$ maintained by the delay attack processes.

Labelled Transition Semantics. A smart home system is modelled as a labelled transition system (LTS) in the structural operational semantics (SOS) style. The transitions are of kind $P \xrightarrow{\alpha} Q$ for actions (a.k.a., labels) ranged over by α in the set $\{\texttt{idle}, \tau, \texttt{snd}\,\bar{c}\langle v\rangle, \texttt{rcv}\,c(x), \texttt{read}\,s(x), \texttt{read}\,a(x), \texttt{wrt}\,\bar{a}\langle v\rangle\}$. The transition rules are shown in Table VIII. Most of them are the same as the standard ones [41], [42], except that we distinguish the transition rules for parallel compositions under delay-absent and delay-present situations where (NoDelayCom) and (DelayCom) are used, respectively. For brevity, we sometimes use the original notations $\texttt{Sys} = E \bowtie P$ to denote a system under delay attacks, without rewriting the attacked channels.

C. Formal Verification and Categorization of DAI Attacks

We use a shorthand $\texttt{Sys} = \texttt{E} \circ \tilde{\texttt{R}}[\tilde{D} \triangleright \tilde{L}]$ to denote a system where each rule $\texttt{R}_j \in \tilde{\texttt{R}}$ run on a platform $L_j \in \tilde{L}$ and reads/writes a set of devices $D_j \in \tilde{D}$. To make a practical sense, we only consider *well-formed* systems where every platform has access to all devices used by the rules running on it, i.e., $D_j \subset (\tilde{\mathcal{S}} \cup \tilde{\mathcal{A}})$. $\texttt{E}$ denotes the physical environment. Consider an attack-present system $\texttt{Sys} = \texttt{E} \circ (\texttt{R1}[(D_1 \triangleright L_1] \parallel \texttt{R2}[(D_2 \triangleright L_2])$ where two automation rules $\texttt{R1}$ and $\texttt{R2}$ are installed on two platforms L_1 and L_2, respectively. L_1 and L_2 can be the same or different platforms. We define a specification system which runs $\texttt{R1}$ and $\texttt{R2}$ on an oracle platform L which accesses devices without any delays, i.e., $\texttt{Sys}^* = \texttt{E} \circ (\texttt{R1}[(D_1 \triangleright L^*] \parallel \texttt{R2}[(D_2 \triangleright L^*])$. $\texttt{Sys}$ and $\texttt{Sys}^*$ describe the real system and the mentally expected system by users, respectively.

Verification in SPSP Systems. A recent work [12] adopts Generalized Non Deducibility on Composition (GNDC) [78] to define a CRI-free system: a rule $\texttt{R1}$ does not interfere with another rule $\texttt{R2}$ if the compositional runtime behavior of $\texttt{R1}$ and $\texttt{R2}$ does not differ from the behavior of $\texttt{R2}$ when running along. Formally, $\texttt{R1}$ does not interfere with $\texttt{R2}$ under a hiding weak bisimulation notion:

$$\texttt{E} \circ (\texttt{R1}[(D_1 \triangleright L] \parallel \texttt{R2}[(D_2 \triangleright L]) \approx_{H_{\texttt{R1}}} \texttt{E} \circ \texttt{R2}[(D_2 \vdash L)] \qquad (1)$$

for $H_{\texttt{R1}} \stackrel{\text{def}}{=} \text{obserable}(\texttt{R1})$ denoting a set of hidden actions of $\texttt{R1}$ that yield to observable results. Two rules $\texttt{R1}$ and $\texttt{R2}$ are CRI-free when $\texttt{R1}$ and $\texttt{R2}$ do not interfere with each other.

However, this only considers an SPSP scenario where all automation rules run on a platform equivalent to an oracle platform L^* and can only detect CRI rooted from mis-programming or mis-configuration, subject to the same limitations of other existing work [5], [6], [7], [9].

Verification in SPMP and MP Systems. In this paper, we aim to model the uncovered CRI scenarios in a situation where two rules automation rules R_1 and R_2 may run on different platforms and the communication channels suffer from delays. In this sense, R_1 and R_2 are CRI-free if in addition to the hiding weak bisimulation, a standard weak bisimulation holds:

$$\texttt{E} \circ (\texttt{R1}[D_1 \triangleright L_1] \parallel \texttt{R2}[D_2 \triangleright L_2]) \approx \texttt{E} \circ (\texttt{R1}[D_1 \triangleright L^*] \parallel \texttt{R2}[D_2 \triangleright L^*]) \qquad (2)$$

Formula 2 ensures that the interactions between $\texttt{R1}$ and $\texttt{R2}$ in the real system and specification system are equivalent at every time unit. However, it is a sufficient but not necessary condition. It is too strict to say that the real system has a significant CRI problem if it only deviates from the specification system at a specific time unit. For example, if the attacker delays all communications evenly by one time unit, the real system will lag behind the specification system but may still produce the correct automation result. Thus, verifying CRI with Formula 2 usually causes false alarms. To address this problem, we define an *idle-insensitive weak bisimulation* $\approx_{\texttt{idle}}$. Let $\Rightarrow_{\texttt{idle}} \stackrel{\text{def}}{=} \xrightarrow{\{\tau, \texttt{idle}\}}{}^{*}$ denote a sequence of zero or more transitions each of which could be a τ or $\texttt{idle}$ transition. We replace the notion of an experiment $\stackrel{e}{\Rightarrow}$ in the definition of standard *weak bisimulation* [41], [79] with $\stackrel{e}{\Rightarrow}_{\texttt{idle}} \stackrel{\text{def}}{=} \Rightarrow_{\texttt{idle}} \xrightarrow{\alpha_1} \Rightarrow_{\texttt{idle}} \cdots \Rightarrow_{\texttt{idle}} \xrightarrow{\alpha_1} \Rightarrow_{\texttt{idle}}$. Thus, the interaction between $\texttt{R1}$ and $\texttt{R2}$ are said to be *CRI-resistant*, if it holds that

TABLE VIII: LTS for processes. $\Rightarrow \stackrel{\text{def}}{=\!=\!\rightarrow}{}^{*}$ denotes a sequence of zero or more τ transitions $\stackrel{\tau}{\rightarrow}$. $\stackrel{\alpha_1 \cdots \alpha_n}{\Longrightarrow} \stackrel{\text{def}}{=\!=\!\Rightarrow} \stackrel{\alpha_1}{\rightarrow} \Rightarrow \cdots \Rightarrow \stackrel{\alpha_1}{\Longrightarrow} \Rightarrow$ denotes interleaving a sequence of α transitions with any number of τ transitions.

$$\text{(Out)}\ \frac{-}{\mathtt{snd}\,\overline{c}\langle v\rangle.P \xrightarrow{\mathtt{snd}\,\overline{c}\langle v\rangle} P} \qquad \text{(TimeNil)}\ \frac{-}{\mathtt{nil} \xrightarrow{\mathtt{idle}} \mathtt{nil}} \qquad \text{(TimePar)}\ \frac{P \xrightarrow{\mathtt{idle}} P'\ \ Q \xrightarrow{\mathtt{idle}} Q'\ \ P \parallel Q \overset{\tau}{\not\rightarrow}}{P \parallel Q \xrightarrow{\mathtt{idle}} P' \parallel Q'}$$

$$\text{(In)}\ \frac{-}{\mathtt{rcv}\,c(x).P \xrightarrow{\mathtt{rcv}\,c(v)} P\{v/x\}} \qquad \text{(WriteAct)}\ \frac{-}{\mathtt{wrt}\,\overline{a}\langle v\rangle.P \xrightarrow{\mathtt{wrt}\,\overline{a}\langle v\rangle} P} \qquad \text{(ReadAct)}\ \frac{-}{\mathtt{read}\,a(x).P \xrightarrow{\mathtt{read}\,a(v)} P\{v/x\}}$$

$$\text{(Par)}\ \frac{P \xrightarrow{\alpha} P'\ \ \alpha \neq \mathtt{idle}}{P \parallel Q \xrightarrow{\alpha} P' \parallel Q} \qquad \text{(Sum)}\ \frac{\pi_1.P \xrightarrow{\alpha} P}{\pi_1.P + \pi_2.Q \xrightarrow{\alpha} P} \qquad \text{(ReadSen)}\ \frac{-}{\mathtt{read}\,s(x).P \xrightarrow{\mathtt{read}\,s(v)} P\{v/x\}}$$

$$\text{(Rec)}\ \frac{P\{\tilde{w}/\tilde{x}\} \xrightarrow{\alpha} Q\ \ H(\tilde{x}) = P}{H\langle\tilde{w}\rangle \xrightarrow{\alpha} Q} \qquad \text{(IfTrue)}\ \frac{[\![b]\!] = \mathtt{true}\ \ P \xrightarrow{\alpha} P'}{\mathtt{if}\,(b)\,\{P\}\,\mathtt{else}\,\{Q\} \xrightarrow{\alpha} P'} \qquad \text{(NoDelayCom)}\ \frac{P \xrightarrow{\mathtt{snd}\,\overline{c}\langle v\rangle} P'\ \ Q \xrightarrow{\mathtt{rcv}\,c(x)} Q'}{P \parallel Q \xrightarrow{\tau} P' \parallel Q'\{v/x\}}$$

$$\text{(Elapse)}\ \frac{-}{\mathtt{idle}.P \xrightarrow{\mathtt{idle}} P} \qquad \text{(Else)}\ \frac{[\![b]\!] = \mathtt{false}\ \ Q \xrightarrow{\alpha} Q'}{\mathtt{if}\,(b)\,\{P\}\,\mathtt{else}\,\{Q\} \xrightarrow{\alpha} Q'} \qquad \text{(DelayCom)}\ \frac{P \xrightarrow{\mathtt{snd}\,\overline{c}\langle v\rangle} P'\ \ Q \xrightarrow{\mathtt{rcv}\,c(x)} Q'\ \ \mathtt{Delay}(c,c',id,val,t)}{P \parallel Q \xrightarrow{\mathtt{idle}^t} P' \parallel Q'\{v/x\}}$$

$$\mathtt{E_O}(\mathtt{R1}[D_1 \triangleright L_1] \parallel \mathtt{R2}[D_2 \triangleright L_2]) \approx_{\mathtt{idle}} \mathtt{E_O}(\mathtt{R1}[D_1 \triangleright L^*] \parallel \mathtt{R2}[D_2 \triangleright L^*]) \quad (3)$$

Provided a specific home deployment and attack strategy (the target event/command), we can use Formula 3 to verify if the system yields different automation results from a specification system, or say if the real system has unique CRI issues under the attack. We can easily prove that the two rules in the running example always generate the correct result in SPSP system even under delay attacks. However, when the two rules run on different platforms, delaying the `UNLOCKED` value on channel $\overline{event}_{\mathrm{B}}$ results in a violation of Formula 3. Due to the space limit, proofs are omitted.

Methodology for Systematic Categorization. To systematically and comprehensively discover possible DAI attacks that cause CRI problems, we generalize a smart home deployment and represent it as the smart home calculus parametric on some enumerable factors (sensor measurements, automation rules, attack strategies). We then enumerate these factors to find violation scenarios of the CRI-free condition (Formula 3). State explosion challenges exist since smart home deployments are highly diverse and it is infeasible to enumerate all deployments and states. To address this challenge, we make several assumptions or abstractions to reduce the complexity of smart home deployments and the enumeration space.

First, we assume that the natural communication delays are negligible compared to the injected delays by the attacker. We omit the sub-processes that only forward messages, such as hubs, endpoint clouds, routers, and add up the communication delays on these channels (if any) to the end-to-end delay between a device and a platform.

Second, we simplify the models of IoT devices and rules. Note that automation rules follow a trigger-condition-action paradigm. A rule's trigger is actually a boolean expression that checks a device value (e.g., when temperature exceeds $75°F$) and its condition is a set of such boolean expressions. A rule action is not always binary (e.g., dim the light to 75%). However, the interaction relation between a rule's action and another rule's trigger, condition, or action is binary. For example, one may ask if a rule `R1`'s action can control an actuator (e.g., turn on lights) to a value that makes another rule `R2`'s trigger (e.g., when the brightness is high) or condition (e.g., if the lights are on) satisfied, or if the action is contradictory to another rule `R2`'s action (i.e., turn off lights). Therefore, we binarize the automation rules and device values. We consider that each device has two values (0 and 1) and each rule deals with binary values of devices. Without loss of generality, we assume each rule condition only checks one device.

Third, we simplify the rule-device bindings. Without a specific home deployment, one cannot know what devices are bound to each rule. In our simplification, three devices are granted to each rule for its trigger, condition and action, respectively. Thus, two rules `R1` and `R2` choose from a set of six devices $D = \{d_1, d_2, \cdots, d_6\}$. In practice, a rule's trigger and condition always check different devices. We pick $D_1 = \{d_1, d_2, d_k | k \in \{1,2,3\}\}$ for `R1` and $D_2 = \{d_i, d_j, d_k | d_i, d_j, d_k \in D; i \neq j; x < y$ if $x > 3, y > 3, x \in \{i,j\}, y \in \{j,k\}\}$ for `R2`. We only consider the interactions between two automation rules `R1` and `R2` since most, if not all, interactions among more than two rules contain sub-interactions between two rules.

With the above abstractions, we enumerate the rules, rule-device bindings, initial device states and the attacker's target events/commands and verify CRI in each configuration. In this process, we observe that the attacks against some different configurations essentially belong to the same category. Therefore, we perform a manual qualitative analysis to combine the similar attack scenarios and classify DAI attacks into seven categories (see Section IV-B). Note that the above abstractions are only for easing the exploration of possible attacks. Provided a specific deployment, our theoretic technique (calculus) can verify CRI without the abstractions.

D. IRB Approval

We have received the approval from the Institutional Review Boards (IRB) in the institution where all experiments are conducted and all apartment residents (undergraduate and graduate students) are affiliated with. The participants were recruited following the IRB protocol and 500 USD was paid to the each testbed. Devices and rules were furnished by the researchers. Participants were informed of the possible outcomes caused by the attacks and the experiments were monitored by the researchers to avoid any hazards. The data collected from both testbeds do not contain personally identifiable information and are stored in a secure way. Only the researchers identified on the IRB protocol have access to the data.