MP-Mediator: Detecting and Handling the New Stealthy Delay Attacks on IoT Events and Commands

Xuening Xu
Stevens Institute of Technology
Hoboken, NJ, USA
xxu64@stevens.edu

Chenglong Fu
Univ. of North Carolina at Charlotte
Charlotte, NC, USA
chenglong.fu@uncc.edu

Xiaojiang Du
Stevens Institute of Technology
Hoboken, NJ, USA
xdu16@stevens.edu

ABSTRACT

In recent years, intelligent and automated device control features have led to a significant increase in the adoption of smart home IoT systems. Each IoT device sends its events to (and receives commands from) the corresponding IoT server/platform, which executes automation rules set by the user. Recent studies have shown that IoT messages, including events and commands, are subject to stealthy delays ranging from several seconds to minutes, or even hours, without raising any alerts. Exploiting this vulnerability, adversaries can intentionally delay crucial events (e.g., fire alarms) or commands (e.g., locking a door), as well as alter the order of IoT messages that dictate automation rule execution. This manipulation can deceive IoT servers, leading to incorrect command issuance and jeopardizing smart home safety. In this paper, we present *MP-Mediator*, which is the first defense system that can *detect* and *handle* the new, stealthy, and widely applicable delay attacks on IoT messages. For IoT devices lacking accessible APIs, we propose innovative methods leveraging virtual devices and virtual rules as a bridge for indirect integration with MP-Mediator. Furthermore, a VPN-based component is proposed to handle command delay attacks on critical links. We implement and evaluate MP-Mediator in a real-world smart home testbed with twenty-two popular IoT devices and two major IoT automation platforms (IFTTT and Samsung SmartThings). The experimental results show that MP-Mediator can quickly and accurately detect the delay attacks on both IoT events and commands with a precision of more than 96% and a recall of 100%, as well as effectively handle the delay attacks.

CCS CONCEPTS

• **Security and privacy** → **Systems security**; • **Computer systems organization** → *Sensor networks.*

KEYWORDS

IoT, security, delay attack, detection, handling

ACM Reference Format:
Xuening Xu, Chenglong Fu, and Xiaojiang Du. 2023. MP-Mediator: Detecting and Handling the New Stealthy Delay Attacks on IoT Events and Commands. In *The 26th International Symposium on Research in Attacks, Intrusions and Defenses (RAID '23), October 16–18, 2023, Hong Kong, China.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3607199.3607225

RAID '23, October 16–18, 2023, Hong Kong, China

ACM ISBN 979-8-4007-0765-0/23/10…$15.00
https://doi.org/10.1145/3607199.3607225

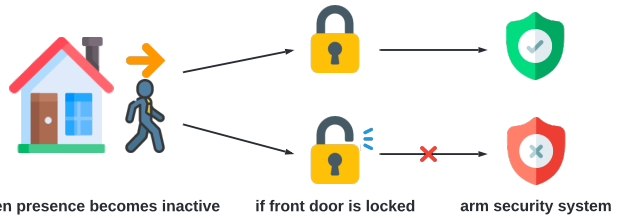


Figure 1: An example of the delay attack: The automation rule, *"When presence becomes inactive, if the front door is locked, then arm the security system"* is maliciously disabled by delaying the *"front door is locked"* event, which leaves the security system disarmed after user's leaving. Even if the *"front door is locked"* event is received later, there is no more *"presence inactive"* event to trigger the automation to arm the security system, leaving the home unprotected.

1 INTRODUCTION

The rapid development of the Internet of Things (IoT) facilitates the popularity of smart home systems. By 2021, smart home IoT has entered more than 43% of U.S. households [55]. In a typical smart home, IoT devices are integrated into IoT platforms such as Samsung SmartThings [14], Apple HomeKit [8], and IFTTT [9], which provide a great convenience for home automation by executing user-customized automation through IoT platform servers.

Despite the benefits of home automation, it also creates new attack surfaces for adversaries. Since the execution of automation rules is driven and fulfilled by messages between IoT devices and servers, the IoT message/communication channels become a major target of cyberattacks against smart home IoT systems. Recent studies [25, 35] present a new delay attack on IoT messages that can stealthily delay IoT events and commands from dozens of seconds to several minutes (or even hours in some cases) without triggering any alarms, even if the IoT messages are protected by the Transport Layer Security (TLS) protocol. Attackers can delay events and commands of safety-critical devices (e.g., smoke detectors, smart locks) or maliciously disable (or trigger) the execution of automation rules by reversing the order of rule-relevant messages. An example of the new delay attack is given in Figure 1.

In the recent study [35], Fu et. al, also briefly discuss some countermeasures to the new delay attack, including shortening message timeout periods and checking event timestamps. They find that these countermeasures are either incapable of handling some attack cases or require modifying the software of IoT devices and

servers. Furthermore, although this vulnerability has been reported to major IoT vendors for more than one year, we are not aware of any mitigation or defense against the new delay attacks on the affected IoT devices, which include the devices from most major IoT vendors, such as Google, Amazon, Samsung, SimpliSafe, Ring, and Apple. The existing IoT security solutions, such as policy-based approaches [24, 31] and anomaly detection systems [20, 21, 36, 54], cannot address the new delay attacks (details are given in Section 7). As a result, it is still an open question regarding how to detect and handle the new delay attack. Without an effective defense method, millions of smart homes are at risk.

Targeting this urgent issue, we present the Multi-Platform Mediator (MP-Mediator) as the first work of defense against the IoT message delay attack. MP-Mediator decouples the direct connections between IoT devices and automation servers and evaluates the messages being exchanged in terms of their latency of being acknowledged. We then conduct a systematic investigation of IoT message delay attacks in different network and device integration scenarios, and propose customized strategies for detecting and handling each of them. The core idea is as follows: MP-Mediator starts a timer and records the time when a new IoT device event is sent out by sniffing the wireless communication in the smart home local network. The time is then used as the ground truth and compared with the time when the message is acknowledged by the receiver to detect delay attacks. After a delay attack is detected, it will be handled by forwarding the delayed events to IoT servers or secretly tunneling the delayed commands to target devices. The details are given in Section 5.

To make the MP-Mediator an implementable solution, there are some prominent challenges.

Challenge 1: Some of the IoT devices cannot be directly accessed due to the lack of APIs, which makes IoT events or commands non-observable to MP-Mediator (or any other security modules). Without observing IoT events or commands, it is impossible to detect any delay attacks on them.

Challenge 2: The delay attacks can also delay traffic between MP-Mediator and IoT platforms without causing any alarms. It is not easy to quickly detect a delay attack on this network link because we do not know the exact time when an IoT platform receives an event or sends out a command.

Challenge 3: In some cases, a command delay attack can block the only path to an IoT device, which makes it unable to receive any commands, even after the command delay attack has been detected.

To tackle the above challenges, we design novel virtual-device and virtual-rule-based approaches, as well as a VPN-based method for detecting and handling the new delay attacks on IoT messages. A ***virtual device*** is a computer program that represents an instance of an IoT device. A ***virtual rule*** is an automation rule that involves virtual devices. For instance, a virtual rule, *When the physical light is turned on, turn on the virtual device Vir-Light*, synchronizes the states between the physical light and the virtual device Vir-Light.

We implement MP-Mediator in a real-world smart home testbed with twenty-two commodity IoT devices and two popular IoT platforms. We evaluate MP-Mediator on both IoT events and commands, and the results show that MP-Mediator can detect the new delay attacks with a precision of over 96% and a recall of 100%, and then effectively handle the delay attacks.

Our **contributions** are summarized as follows:

- This is the first work that systematically studies the countermeasures (including both ***detection*** and ***handling***) for the stealthy delay attacks on IoT events and commands. We categorize network links between IoT devices and IoT servers for different connection scenarios, and we consider various attack strategies on all the network links.
- To address **Challenge 1**, we propose to create ***virtual device*** for each physical IoT device in a smart home that does not have APIs for accessing it. A virtual device is synchronized with its corresponding real device through some ***virtual rules***. Virtual devices and virtual rules enable the MP-Mediator to indirectly observe IoT devices that do not have APIs for accessing them. Details about virtual devices and virtual rules can be found in Section 5.2.2.
- To address **Challenge 2**, we propose using virtual devices and virtual rules to create short-interval heartbeats between MP-Mediator and IoT platforms, which enables quick detection of the delay attacks (see Section 5.2.4 and Section 5.2.5).
- To address **Challenge 3**, we propose a VPN-based method to tunnel a newly established connection such that a command can be re-sent via the VPN tunnel without being noticed by the attacker. Details are given in Section 5.3.1.
- We implement and evaluate MP-Mediator in a real-world smart home testbed with 22 different IoT devices and two popular IoT integration platforms, which shows the effectiveness of MP-Mediator.

The rest of the paper is organized as follows. We describe the background in Section 2 and discuss the attack model in Section 3. We present the system overview in Section 4. We detail the system design of detecting and handling the delay attacks in Section 5. In Section 6, we evaluate the performance of our approaches. We discuss related work in Section 7. Section 8 presents additional discussions. Finally, we conclude this work in Section 9.

2 BACKGROUND

We give the background of smart home systems in Section 2.1, and discuss related work of identifying devices, events, and commands by analyzing encrypted traffic in Section 2.2. The new delay attacks are presented in Section 2.3.

2.1 Smart Home Systems

A typical smart home contains several components as shown in Figure 2: IoT devices, hubs, and IoT servers.

IoT Devices and Hubs. IoT devices deployed in a smart home employ various wireless communication technologies, such as Wi-Fi, Zigbee, and Z-Wave. Wi-Fi devices can be directly connected to a home router, while Zigbee/Z-Wave devices require a dedicated hub as the gateway to connect Zigbee/Z-Wave networks with the home Wi-Fi network, converting non-IP payloads to IP-based payloads and then forwarding them to the home Wi-Fi router.

IoT Servers. After receiving events from IoT devices, a Wi-Fi router sends them to IoT platforms/servers, which include local servers and cloud servers based on the location. Local IoT servers (i.e., local platforms), such as HomeKit, run on a local device, while cloud IoT servers are hosted on cloud servers. Further, cloud servers

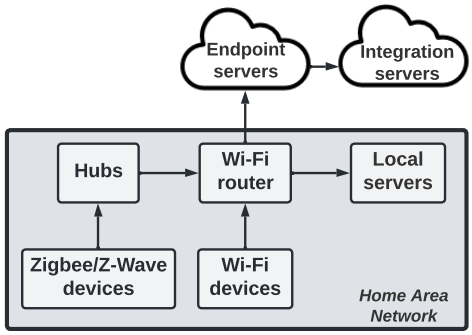


Figure 2: System model of a typical smart home.

can be divided into endpoint servers and integration servers (i.e., integration platforms). An endpoint server is also called a *vendor cloud server* that is operated by a specific IoT device vendor to only support the interaction with its own IoT devices. An *integration platform* aims to integrate various IoT devices from different vendors. Unlike endpoint servers that directly interact with their devices or hubs, an integration platform controls devices via their vendor cloud servers using cloud-to-cloud communications.

2.2 Analyzing Encrypted Traffic

Recent studies have shown the feasibility and effectiveness of inferring information about a smart home by analyzing its encrypted traffic. Specifically, packet information, such as MAC/IP addresses, packet lengths, DNS, etc., can be used to identify IoT devices [29, 48, 53, 60, 61], as well as recognize device events and commands in a real-time manner [18, 57, 62]. Furthermore, automation rules can also be inferred by analyzing a sequence of successfully recognized events and commands [40, 45, 56, 62]. Such inference has a high accuracy of 91% in identifying IoT devices [61] and 97% on average in recognizing device events [57]. Besides, recent work [62] achieves high precision in inferring automation rules from the encrypted traffic. In this work, we assume attackers can exploit the above information to infer smart home information to launch the delay attacks. Besides, the proposed defense system can sniff the local traffic and make use of the inferred events and commands to detect and handle the delay attacks.

2.3 Delay Attacks on IoT Events and Commands

Recent works [25, 35] reveal a **new delay attack** on almost all commercial smart home IoT devices. The new delay attack is different from jamming attacks or other DoS and MITM attacks in the following aspects: 1) it does not cause any alarm in any layer of the IoT network protocol stack; 2) a delayed message is eventually delivered (after the delay caused by the attack), so the receiver does not see any problem. According to [35], the delay on IoT events or commands ranges from dozens of seconds to hours, depending on the brand/type of IoT devices and the cloud. The new delay attack utilizes the fact that timeout detection in the TCP layer is decoupled from data protection in the TLS layer. The attack can

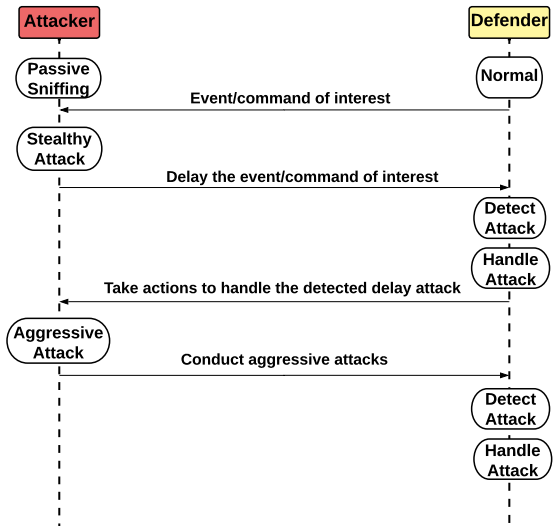


Figure 3: Interactions between the attacker and the defender.

be successfully launched without knowing the TLS session keys or causing any TLS alerts.

Fu et al. present three types of delay attacks in [35], including the state-update delay attack, the action delay attack, and the erroneous execution attack. Delaying IoT event or command messages could cause severe damage to the safety of users or property. Below are some examples in the form of [delaying events → (could cause) consequence]: [delaying water leak sensor events → flooding], [delaying smoke detector events → fire], [delaying camera/contact sensor events → home invasion undetected], [delaying carbon monoxide sensor events → CO high/death], [delaying open window command → CO high], [delaying start water sprinkler command → fire]. The consequences caused by the delay attacks could be disastrous or even fatal to the residents. Hence, it is a pressing matter to design effective detection and handling methods for the new delay attacks.

3 ATTACK MODEL

The goal of the attacker is to induce safety risks in the victim's smart home by delaying IoT events and/or commands of target IoT devices. An attacker can eavesdrop on the traffic of a smart home. The attacker can use a compromised device or a controlled device (e.g., by deploying his own device nearby) to hijack TCP sessions, and then proactively delay events and commands sent over the hijacked TCP session. Note that the TCP hijacking attack can only relay traffic but cannot modify payloads of network packets due to the protection provided by TLS. In addition, Zigbee IoT devices in a smart home may form a mesh network. Also, some homes may use a Wi-Fi range extender. An attacker may exploit a relay node in a mesh network or a Wi-Fi range extender to launch the delay attacks. As discussed in Section 2.2, an attacker has the capability to identify devices and message types from encrypted traffic. This gives an attacker the ability to delay a specific event or command on a TLS-protected TCP session, but the attacker cannot modify the payloads. **Similar to the assumptions in IoTGuard [24], IoTSafe [31] and**

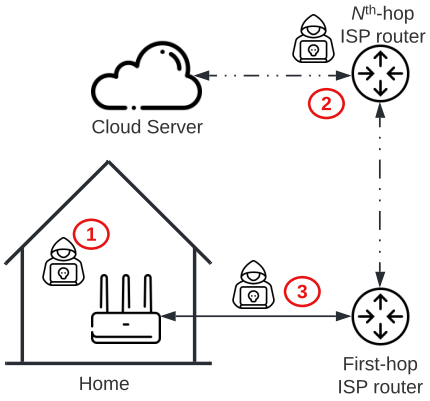


Figure 4: Three methods to launch delay attacks at three different locations: ① LAN delay attack; ② specific-cloud delay attack; ③ first-hop delay attack.

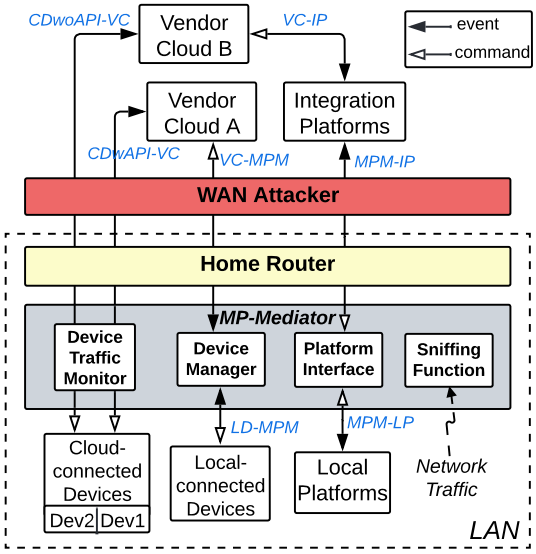


Figure 5: Architecture of MP-Mediator. Devices on a) *CDwAPI-VC*: cloud APIs; b) *CDwoAPI-VC*: no APIs available; c) *LD-MPM*: local APIs or Zigbee/Z-Wave. Dev1 and Dev2 are cloud-connected devices on *CDwAPI-VC* and *CDwoAPI-VC*, respectively. Three different event flows (solid arrows) are detailed below: 1) Cloud-connected Device (Dev1) → Device Traffic Monitor → *CDwAPI-VC* → Vendor Cloud A → *VC-MPM* → Device Manager → Platform Interface → *MPM-IP* / *MPM-LP* → Integration Platform / Local Platform; 2) Cloud-connected Device (Dev2) → Device Traffic Monitor → *CDwoAPI-VC* → Vendor Cloud B → *VC-IP* → Integration Platform; 3) Local-connected Device → *LD-MPM* → Device Manager → Platform Interface → *MPM-IP* / *MPM-LP* → Integration Platform / Local Platform. The direction of command flows (hollow arrows) is reverse to the event flow directions. The full names of the network link acronyms are given in Table 1.

ContexIoT [43], MP-Mediator itself is assumed to be secure. Also, there are existing approaches and tools (e.g., ArpON [50], XArp [47] and DefendARP [52]) that can protect MP-Mediator from ARP spoofing attacks. I.e., MP-Mediator is resistant to MITM and redirect attacks, and MP-Mediator can successfully send messages to local devices without being affected by the delay attacks. Note that due to the hardware and power constraint of low-cost IoT devices, it is impractical to deploy such approaches and tools on the IoT devices to protect them from MITM and redirect attacks. The virtual devices proposed by us are part of the MP-Mediator software and run on MP-Mediator. Therefore, the virtual devices are secure and cannot be compromised by attackers.

As shown in Figure 3, an attacker can passively sniff the traffic first. When he sniffs any event or command of interest, the attacker intends to keep his attacks stealthy (i.e., not being detected), but the attacker may become aggressive if he finds his attacks have been detected (i.e., seeing the defender is taking action to handle the attacks). Based on the location where the attacker launches the delay attacks (Figure 4), there are three different delay attack methods with corresponding capabilities as follows:

Method 1: LAN Delay Attack. A LAN delay attacker either compromises an existing IoT device in the victim's home or deploys a Wi-Fi device near the victim's home, and thus has control of the device. With the controlled device, the attacker can sniff Wi-Fi traffic and hijack TCP sessions (e.g., the well-known ARP spoofing attacks [58, 59]) of the target device. Then the attacker can selectively delay a specific event or command of the target device, which is inferred from the encrypted traffic. If being detected, an aggressive LAN attacker may try to hijack all TCP sessions to block the communications of the entire smart home.

Method 2: Specific-cloud Delay Attack. A specific-cloud delay attacker compromises an ISP router between the home router and a specific IoT cloud server. Then the attacker can see all the traffic between the home router and that specific IoT cloud server, but he cannot see traffic between the home router and other cloud servers. This type of attacker can hijack TCP sessions of IoT devices that are connected to the target cloud server and delay the IoT devices' events or commands at will. If being detected, an aggressive specific-cloud attacker could block all traffic between the home router and the cloud server, causing all connected devices to lose connection. For each individual device, the consequence is the same as that of the stealthy delay attacks. Thus, we do not further discuss aggressive specific-cloud attackers.

Method 3: First-hop Delay Attack. A first-hop delay attacker compromises the connection between the home router and the first-hop ISP router, which allows the attacker to see all network traffic between the home and the Internet. For example, an attacker can find the network cable outside the victim's home and deploy an active Test Access Point (TAP) [38] on the cable so that he can hijack all TCP sessions. Besides, the attacker can identify IoT devices and messages using existing techniques as discussed in Section 2.2, and then he can delay any events/commands to/from cloud servers. If being detected, an aggressive first-hop attacker could block all network traffic from/to the victim's home, which results in a loss of Internet connection.

Table 1: Descriptions of link acronyms in Figure 5.

Link Acronym	Full Description
CDwAPI-VC	The network link between Cloud-connected Device with APIs and Vendor Cloud.
CDwoAPI-VC	The network link between Cloud-connected Device without APIs and Vendor Cloud.
VC-MPM	The network link between Vendor Cloud and Multi-Platform Mediator.
LD-MPM	The network link between Local-connected Device and Multi-Platform Mediator.
MPM-LP	The network link between Multi-Platform Mediator and Local Platform.
MPM-IP	The network link between Multi-Platform Mediator and Integration Platform.
VC-IP	The network link between Vendor Cloud and Integration Platform.

4 SYSTEM OVERVIEW

MP-Mediator is a defense system that can effectively detect and handle IoT message delay attacks in a smart home ecosystem. MP-Mediator locates between IoT devices and integration platforms. It breaks each direct connection into two separate ones so that IoT messages flow through MP-Mediator where the detection and handling are performed.

4.1 Overview of MP-Mediator

Similar to [30, 42, 46], MP-Mediator can be deployed as software in a Wi-Fi router (e.g., OpenWrt-based router [12]) at home. Figure 5 shows the architecture of MP-Mediator, which includes Device Traffic Monitor, Device Manager, Platform Interface and the sniffing function. Table 1 shows the full descriptions of the link acronyms. Hollow arrows represent event flows and solid arrows represent command flows. We discuss each module below: (a) the Device Traffic Monitor intercepts connections between cloud-connected IoT devices and their vendor cloud servers. It relays network traffic without modifying any packets; (b) the Device Manager is for handling the joining or leaving of devices, as well as receiving events from devices and sending commands to devices; (c) the Platform Interface interacts with cloud platforms, including authorization, authentication, receiving commands from platforms, and forwarding IoT events to platforms; (d) the built-in sniffing function is used to infer device events and commands from the sniffed traffic in LAN. Note that the existing works have achieved satisfying accuracy in inferring IoT events and commands as discussed in Section 2.2.

In addition to the four modules shown in Figure 5, MP-Mediator also has a VPN client and a table built into it, where (e) the VPN client is used to create a VPN tunnel that can secretly forward messages without being discovered by attackers, which is the VPN-based method that is used for handling command delay attacks (more details in Section 5.3.1); (f) the table is maintained by MP-Mediator for storing the status of devices (including virtual devices that are used in virtual rules), as well as access tokens for interfacing with integration platforms, and it is updated based on every received device event (the initial status values in the table are automatically configured by MP-Mediator without any user involvement). With the device table in MP-Mediator, we propose to further confirm the correctness of the sniffed events and commands in the sniffing function, by utilizing the device status stored in the table to improve the accuracy. Taking a smart switch as an example, assume its current status is "OFF" and it is the same as the record in the table. Then the sniffing function inside MP-Mediator sniffs an event from the smart switch in LAN and recognizes it as an "ON" event. By checking the previous status of the smart switch in the table, which is "OFF", we can confidently think that the sniffed event is indeed an "ON" event. Note that the sniffing function is used to infer an event that occurs in LAN before MP-Mediator actually receives it. The table updates the device status based on the actually received event (if not delayed), instead of the sniffed or inferred event.

4.2 Device Traffic Monitor

The Device Traffic Monitor is built inside MP-Mediator to intercept traffic between devices and their vendor cloud servers on *CDwAPI-VC* and *CDwoAPI-VC*, so that it can monitor and control the traffic whenever it is necessary. Specifically, the Device Traffic Monitor listens on a socket on the device side, and when it receives a TCP connection request from a device to a cloud server, the Device Traffic Monitor accepts it and then starts another TCP connection with the cloud server. After both TCP connections are established, TCP payloads are forwarded between them. Although the Device Traffic Monitor intercepts a TCP connection (i.e., it sets up two new TCP connections: one with the device and the other with the cloud), it keeps the original encrypted end-to-end TLS session and other upper layers unchanged.

***CDwAPI-VC*:** IoT devices on *CDwAPI-VC* do not have local APIs and cannot be directly connected to the Device Manager without going through their vendor cloud servers. However, they have cloud APIs available that can be used by the Device Manager to connect these IoT devices.

***CDwoAPI-VC*:** IoT devices on *CDwoAPI-VC* neither have local APIs nor cloud APIs, and thus there is no way to connect them to the Device Manager. These devices can only be connected to their vendor cloud servers.

***VC-IP*:** IoT devices connected to their vendor cloud servers via *CDwoAPI-VC* can be further connected to integration platforms to work with other devices from different vendors.

4.3 Device Manager

MP-Mediator needs to connect IoT devices to manage their events and commands. The Device Manager is responsible for interactions with various IoT devices that use different communication technologies. It receives events and forwards commands to target IoT devices, as illustrated by the arrows in Figure 5. The Device Manager either directly connects IoT devices via *LD-MPM* or connects IoT devices through their vendor cloud servers via *VC-MPM*.

***VC-MPM*:** The Device Manager uses cloud APIs to access devices on their vendor cloud servers. Besides, the Device Manager periodically polls device status from the vendor cloud servers using cloud APIs to keep knowledge of device status up-to-date.

***LD-MPM*:** Devices on *LD-MPM* are directly connected to the Device Manager, which implies that these devices can be locally controlled without going through vendor cloud servers. Such devices include Wi-Fi devices with local APIs, Zigbee devices, and Z-Wave devices.

4.4 Platform Interface

By using platforms' SDKs, after a successful authorization and authentication process, the Platform Interface provides integration platforms access to the instances of the devices that are connected to the Device Manager. Contrary to the Device Manager, it receives commands from and forwards device events to the integration platforms, which can be divided into local integration platforms and cloud integration platforms according to where they are hosted.
***MPM-LP*:** The Platform Interface interacts with local integration platforms within the home LAN, such as HomeKit.
***MPM-IP*:** The Platform Interface interacts with cloud integration platforms, such as SmartThings and IFTTT.

5 MP-MEDIATOR: DETECTING AND HANDLING OF DELAY ATTACKS

We first analyze three different delay attack methods and discuss the relationship between the first-hop attack and the specific-cloud attack in Section 5.1. Then we present detailed solutions to detect and handle the delay attacks on different links based on the architecture of MP-Mediator in Section 5.2 and Section 5.3, respectively. The main idea is to use a timer and the ground truth of the timing of sending events (or receiving commands) from the sniffing function to check if an event or command is successfully delivered within the normal time frame. Section 5.4 discusses the defense against aggressive attacks. Implementation details are given in Appendix A due to the page limit.

5.1 Analysis of Three Attack Methods

5.1.1 LAN Delay Attack. Different from other components, the directly connected devices (on *LD-MPM*) and local platforms (on *MPM-LP*) are only vulnerable to the ***LAN delay attack***, which hijacks the TCP connection between two devices in LAN. Since MP-Mediator is secure and also resistant to both MITM and redirection attacks as assumed in Section 3, the messages sent by MP-Mediator are not affected by the delay attacks. Specifically, commands on *LD-MPM* and events on *MPM-LP* cannot be delayed by LAN delay attacks. For other links, the LAN delay attack is similar to the other two attack methods explained below.

5.1.2 Specific-cloud Delay Attack. The ***specific-cloud delay attack*** hijacks the TCP connection between the home router and a specific cloud server. As shown in Figure 5, the attacker can hijack TCP connections on different links to delay events or commands: (1) *CDwoAPI-VC*, (2) *CDwAPI-VC*, *VC-MPM*, and (3) *MPM-IP*. An attacker could also launch delay attacks on *VC-IP* between two cloud servers. Considering that the traffic on *VC-IP* is composed of event and command messages from many different smart homes, it is difficult for an attacker to precisely delay an event or command of interest for a specific smart home. A Distributed Denial-of-Service (DDoS) attack targeted at a vendor cloud server on *VC-IP* may achieve similar consequences to the delay attacks, which may bring down the entire vendor cloud server and affect all smart homes associated with it. Note that most current cloud servers are well protected against DDoS attacks by various defenses, such as Content Delivery Networks and load balancers. Based on the above, the delay attack on *VC-IP* is not within the scope of this paper.

5.1.3 First-hop Delay Attack. Unlike the specific-cloud attacker, who has control of TCP connections to only one cloud server, an attacker that launches the ***first-hop delay attack*** has control of all network traffic of the victim's home. Besides, the attacker can distinguish different TCP connections and delay multiple TCP connections to various cloud servers. It is possible for a first-hop attacker to delay events/commands on *CDwAPI-VC*, *VC-MPM*, and *CDwoAPI-VC* simultaneously. Note that a cloud-connected device is either connected via *CDwoAPI-VC* or *CDwAPI-VC*. Delaying events/commands on *CDwoAPI-VC* will not affect devices connected via *CDwAPI-VC* and *VC-MPM*, and vice versa. Thus, for each individual cloud-connected device, the first-hop delay attack is similar to the specific-cloud delay attack in terms of consequences, and the defenses for the specific-cloud attack also apply to the first-hop delay attack.

5.2 Detection of Delay Attacks

In this subsection, we present detailed methods for detecting the message delay attack on various links. Although the detection methods are presented separately for each link, they can be used in combination when an attacker delays multiple links of an event/command.

5.2.1 Detecting Delay Attacks on CDwAPI-VC & VC-MPM. Suppose a cloud-connected physical device (e.g., a device in the bottom-left of Figure 5) named "Dev1" is first connected to its vendor cloud via *CDwAPI-VC* and then connected to the Device Manager via *VC-MPM*. The Device Manager receives its events and the Platform Interface forwards them to the integration platform via *MPM-IP*. Any command sent from the integration platform first arrives at the Platform Interface and is then forwarded to the vendor cloud of Dev1 via *VC-MPM* by the Device Manager. Finally, the command is sent to Dev1 through *CDwAPI-VC*.
Against Event Delay Attacks. Although the attacker can choose to delay an event either on *CDwAPI-VC* or *VC-MPM*, essentially the results are the same, i.e., the Device Manager does not receive the event timely. Every time Dev1 sends an event to the integration platform, the sniffing function sniffs the event in LAN and a timer is started. The Device Manager expects to receive the event from *VC-MPM* within a short time. If the timer expires before receiving the event, it indicates that an event delay attack happens on either *CDwAPI-VC* or *VC-MPM* and MP-Mediator detects it.
Against Command Delay Attacks. There are two options to delay commands for Dev1. One is to launch delay attacks on *VC-MPM*, and the other is to delay commands on *CDwAPI-VC*. All commands are initially issued by an integration platform. Every time the Device Manager forwards a command to the target device via *VC-MPM* and then *CDwAPI-VC*, a timer is started to sniff the command in LAN. If the timer expires and the command is not sniffed, MP-Mediator knows the command has been delayed on either *CDwAPI-VC* or *VC-MPM*. To further determine which link has been delayed, we need to know the time when the vendor cloud receives the command. Specifically, if the vendor cloud receives the command before the timer expires, then most likely the command delay attack happens on *CDwAPI-VC*. Otherwise, the attack happens on *VC-MPM*. If an IoT vendor provides such timing information (actually, Samsung SmartThings supports that), MP-Mediator can more precisely determine if a delay attack happens on either *CDwAPI-VC* or *VC-MPM*.

5.2.2 Detecting Delay Attacks on CDwoAPI-VC. Suppose a physical device named "Dev2" connects to its vendor cloud and then to an integration platform via *CDwoAPI-VC* and *VC-IP*. Note that the traffic of Dev2 does not go through MP-Mediator and MP-Mediator cannot directly interfere with its events and commands. To be involved with Dev2, MP-Mediator creates two virtual devices named "Vir-Dev2-E" (for events) and "Vir-Dev2-C" (for commands), which are hosted on MP-Mediator and connected to integration platforms via *MPM-IP*. With the help of virtual devices, MP-Mediator is able to detect (Section 5.2.2) and handle (Section 5.3.2) the delay attacks.

Against Event Delay Attacks. In order to know if an event generated by Dev2 has been received by the integration platform timely, a virtual rule is generated for Dev2 and Vir-Dev2-E: *Turn on/off Vir-Dev2-E if Dev2 is turned on/off*, which establishes a one-to-one correspondence between the physical device Dev2 and the virtual device Vir-Dev2-E. This virtual rule is triggered by Dev2's event to send a command to Vir-Dev2-E on MP-Mediator, so that once the Platform Interface receives a command to Vir-Dev2-E, MP-Mediator knows a Dev2's event has been successfully received by the integration platform. The reason the command to Vir-Dev2-E can be sent to the Platform Interface without being delayed is due to the heartbeat mechanism on *MPM-IP*, which is explained in Section 5.2.5. Every time Dev2 sends an event to its vendor cloud and then to the integration platform, the sniffing function inside MP-Mediator sniffs the event in LAN and a timer is set to wait for the command to be delivered to Vir-Dev2-E. If the timer expires before Vir-Dev2-E receives the command, MP-Mediator considers that the event has been delayed by an attacker.

Against Command Delay Attacks. Solutions for detecting command delay attacks on *CDwoAPI-VC* require knowledge of automation rules. Note that the rule extraction has been widely studied by recent literature and several effective techniques have been proposed, such as code analysis [34, 43] and symbolic execution [23, 26]. For each automation rule that sets an action on Dev2, e.g., *Turn on/off Dev2 if motion is detected when the owner is home*, a virtual rule for Vir-Dev2-C is generated on the integration platform: *Turn on/off Vir-Dev2-C if motion is detected when the owner is home*, where the trigger and condition of the two rules are identical so that they can be triggered at the same time. Every time the Platform Interface receives a command to Vir-Dev2-C, MP-Mediator knows a command is being sent to Dev2 from the integration platform and it starts a timer to wait for the sniffing function to confirm the successful delivery of the Dev2's command by sniffing it in LAN. If the timer expires before sniffing the command, a command delay attack is considered to have occurred on *CDwoAPI-VC*.

5.2.3 Detecting Delay Attacks on LD-MPM. Recall that MP-Mediator is resistant to redirect attacks (Section 3), hence commands from MP-Mediator on *LD-MPM* cannot be delayed. For *LD-MPM*, we only need to detect the event delay attacks. When the sniffing function sniffs an event from a device on *LD-MPM*, a timer is set for the Device Manager to receive it. If the timer expires before receiving the event, then it is considered to be delayed by an attacker.

5.2.4 Detecting Delay Attacks on MPM-LP. As mentioned in Section 5.1, device events from MP-Mediator on *MPM-LP* cannot be delayed by LAN delay attacks due to the resistance of MP-Mediator to redirect attacks (see Section 3). We utilize this feature to maintain simulated heartbeats with a short time interval between MP-Mediator and a local platform (e.g., the Apple HomePod, which hosts the HomeKit platform). Specifically, we create two virtual devices named "Vir-Dev3-HB" and "Vir-Dev3-ACK". In addition, a virtual rule is generated on the local platform: *Turn on/off Vir-Dev3-ACK when Vir-Dev3-HB is turned on/off*. MP-Mediator periodically toggles Vir-Dev3-HB in a short interval to generate device events to simulate *HeartBeat* messages, which trigger the virtual rule on the local platform to issue commands to Vir-Dev3-ACK. The commands act as acknowledgment messages to the heartbeats. In this way, the short-period event and command messages simulate heartbeat and acknowledgment messages, which check the connection of *MPM-LP* at short intervals and detect possible delay attacks. Note that an attacker does not gain anything by delaying heartbeat or acknowledgment messages. A delayed event or command on *MPM-LP* also delays subsequent heartbeat and acknowledgment messages in the same session. Thus, any delay attacks on *MPM-LP* can be detected by the simulated heartbeat mechanism. If MP-Mediator has not received command messages to Vir-Dev3-ACK for a while, it considers that a command delay attack has happened on *MPM-LP*. With the heartbeats, a delay attack can be quickly detected and cannot stay stealthy.

5.2.5 Detecting Delay Attacks on MPM-IP. The method for detecting the delay attacks on *MPM-IP* is similar to that on *MPM-LP* because they are both connections between MP-Mediator and IoT platforms. MP-Mediator creates two virtual devices named "Vir-Dev4-HB" and "Vir-Dev4-ACK", and connects them to the integration platform. Similarly, a virtual rule is also generated to realize the interaction between these two virtual devices. MP-Mediator periodically toggles Vir-Dev4-HB in a short time interval to simulate *HeartBeats* between MP-Mediator and the integration platform. Unlike heartbeat packets on *MPM-LP* that are completely within the LAN, the heartbeat packets on *MPM-IP* could be delayed by attackers because *MPM-IP* actually consists of multiple hops that are outside the LAN (e.g., ISP routers). If a heartbeat packet is missing, then MP-Mediator detects there is a delay attack on *MPM-IP*. With a short heartbeat interval, an attacker cannot launch the *stealthy* delay attacks on *MPM-IP*. If an attacker insists on launching delay attacks on *MPM-IP*, it can be quickly detected.

5.3 Handling Delay Attacks

In this section, we present methods for handling the delay attacks on different links, using the same examples and descriptions from Section 5.2. Similar to the detection methods, the handling methods can also be used in a combined manner for handling delay attacks on multiple links.

5.3.1 Handling Delay Attacks on CDwAPI-VC & VC-MPM.

Against Event Delay Attacks. During the automation rule configuration stage, a user determines the urgency of the rules using either common sense (e.g., a fire alarm is urgent) or online/expert recommendations. If an event is delayed, MP-Mediator first examines whether the delayed event is urgent. If urgent (e.g., smoke/fire detected), the Device Manager directly sends the corresponding command that should be triggered by the event to the target device.

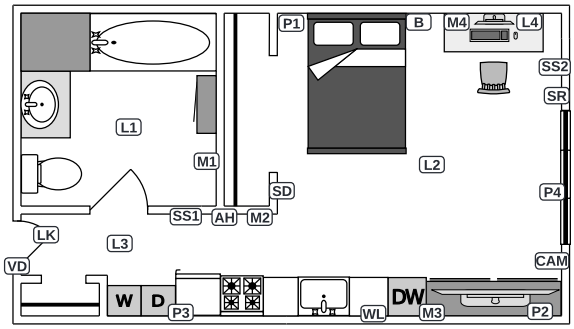


Figure 6: The testbed floor plan and device placement layout. The device abbreviation labels are given in Table 2.

If not urgent, MP-Mediator alerts the user and forwards the sniffed event to the integration platform.

Against Command Delay Attacks. To handle command delay attacks on *VC-MPM*, one option is that MP-Mediator asks the user to issue the command on her smartphone, which uses a cellular network (not subject to the delay attacks) and can successfully deliver the command to the vendor cloud. If the command delay attack happens on *CDwAPI-VC*, then the best one (including MP-Mediator) can do is to alert the user about the attack since the only path to send commands to Dev1 is being attacked. An optional solution is to use the following **VPN-based** method.

MP-Mediator has a built-in OpenVPN client and a VPN-based function to send a delayed command to the target device. For instance, suppose a command that should be sent to Dev1 has been delayed on *CDwAPI-VC*. Recall that the Device Traffic Monitor has control of the current TCP session (Section 4.2), and it uses two different sockets to communicate with the IoT device and the cloud server, separately. To force the IoT device to re-establish a new connection with its cloud server, the Device Traffic Monitor intentionally shuts down the device-side socket so that the IoT device thinks its cloud server terminates the current TCP session and it immediately tries to establish a new TCP session. Meanwhile, MP-Mediator creates a VPN tunnel between Dev1 and its vendor cloud, which encrypts the entire original IP packets, including the original source/destination IP addresses and the original port numbers, which are utilized by attackers to distinguish different connections between IoT devices and vendor cloud servers. The VPN tunnel is completely concealed from the attackers. Once the new TCP session is established, MP-Mediator can re-send the delayed command to the target device Dev1 through the secret VPN tunnel.

5.3.2 Handling Delay Attacks on CDwoAPI-VC.

Against Event Delay Attacks. Upon detecting an event being delayed, MP-Mediator examines if the delayed event is urgent. If urgent, the Device Manager directly sends the corresponding command that should be triggered by the event to the target device. If not urgent, MP-Mediator alerts the user about the detected delay attack. Unlike the handling on *CDwAPI-VC* and *VC-MPM*, MP-Mediator cannot forward the sniffed event to the integration platform since the event does not go through it.

Table 2: IoT devices used in the smart home testbed.

Abbr.	Device Name	Link	Protocol	Attributes
L1, L2	Yeelight light bulb	*LD-MPM*	Wi-Fi	switch
L3, L4	Philips Hue light bulb	*LD-MPM*	Zigbee	switch
P1	Wemo smart plug	*LD-MPM*	Wi-Fi	switch
P2, P3	Kasa smart plug	*LD-MPM*	Wi-Fi	switch
P4	Aqara smart plug	*CDwoAPI-VC*	Zigbee	switch
M1	Aqara motion sensor	*CDwoAPI-VC*	Zigbee	motion
M2	SmartThings motion sensor	*LD-MPM*	Zigbee	motion
M3	Blue by ADT motion sensor	*CDwoAPI-VC*	Z-Wave	motion
M4	Philips Hue motion sensor	*LD-MPM*	Zigbee	motion
B	SmartThings button	*LD-MPM*	Zigbee	button
LK	Kwikset Kevo Lock + Kevo Plus	*CDwAPI-VC*	Wi-Fi	lock
VD	Ring Video Doorbell 4	*CDwAPI-VC*	Wi-Fi	motion, camera, security system
SD	First Alert smoke detector	*LD-MPM*	Z-Wave	smoke
SR	Aeotec Siren 6	*LD-MPM*	Z-Wave	alarm
AH	Aqara hub (with night light)	*CDwoAPI-VC*	Wi-Fi	switch
WL	SmartThings leak sensor	*LD-MPM*	Zigbee	water
CAM	Eufy indoor camera	*CDwAPI-VC*	Wi-Fi	motion, camera
SS1	SimpliSafe security system	*CDwAPI-VC*	Wi-Fi	security system
SS2	Blue by ADT security system	*CDwoAPI-VC*	Wi-Fi	security system

Against Command Delay Attacks. If a command is delayed on *CDwoAPI-VC*, which is the only path to send commands to Dev2, MP-Mediator can only alert users about the command delay attack. Similar to the handling method of the command delay attacks on *CDwAPI-VC* and *VC-MPM*, the user can choose to enable the VPN-based function. If enabled, once the new TCP session is established, MP-Mediator utilizes the one-to-one correspondence built by virtual devices and virtual rules to send the delayed command from the integration platform to Dev2 via the concealed VPN tunnel.

5.3.3 Handling Delay Attacks on LD-MPM. Recall that commands on *LD-MPM* cannot be delayed. We only need to handle event delay attacks, which are similar to the handling methods on *CDwAPI-VC* and *VC-MPM* in Section 5.3.1.

5.3.4 Handling Delay Attacks on MPM-LP & MPM-IP. Attackers would not launch delay attacks on *MPM-LP* or *MPM-IP* if they want to stay stealthy. In case a delay attack is detected on these two links, MP-Mediator immediately sends a notification to inform the user about the delay attack.

5.4 Defense Against Aggressive Attacks

As mentioned in Section 3, the consequence of an aggressive specific-cloud delay attack is the same as that of the stealthy delay attack for each individual device. However, it is different for the LAN delay attack and the first-hop delay attack. If a LAN delay attacker becomes aggressive and tries to hijack all TCP sessions, MP-Mediator will not be able to receive messages from other devices. However, outgoing messages from MP-Mediator cannot be delayed by the LAN delay attack due to its resistance to redirect attacks, and MP-Mediator can still send the user an alert. If a first-hop attacker becomes aggressive and blocks all network traffic, none of the IoT devices can work normally and MP-Mediator loses the ability to send alerts to the user. To deal with this extreme case, MP-Mediator

Table 3: Automation rules installed in the testbed.

Index	Platform	Automation Rules
R1	IFTTT	When the smartphone presence sensor becomes inactive, if **LK** is locked, set **SS1, SS2** to Away and arm **VD**.
R2	SmartThings	When the smartphone presence sensor becomes inactive, if **LK** is locked, turn on **CAM**.
R3	IFTTT	When **M1** detects motion in the bathroom, turn on **L1**.
R4	IFTTT	When **M2** detects motion in the hallway, turn on **AH**.
R5	SmartThings	When **M4** detects motion, turn on **L4**.
R6	IFTTT	When the smartphone presence sensor becomes inactive, turn off **L1, L2, L3, L4** and **P1, P2, P3, P4**.
R7	SmartThings	When the smartphone presence sensor becomes active, unlock **LK**.
R8	SmartThings	When **SD** detects smoke, sound **SR** and turn on **L2**.
R9	IFTTT	When **WL** detects water leak, if **M3** detects no motion, turn off **P3** (simulate water valve).
R10	IFTTT	When **B** is pressed, turn off **L1, L2, L3, L4**.

can use the heartbeat connection on *MPM-IP* and generate a virtual rule on the integration platform: *Alert the user if Vir-Dev4-HB has not been turned on/off (i.e., no heartbeat) for more than 30 seconds*. In this way, a user can still receive alert messages from the integration platform when an aggressive attack happens and all devices at home (including MP-Mediator) lose Internet connection.

6 EVALUATION

In Section 6.1, we introduce the smart home testbed that is deployed in the real world to evaluate the proposed MP-Mediator. In Section 6.2, we present the time needed for MP-Mediator to get notified about the successful delivery of an event or command in normal cases without the delay attacks. Section 6.3 presents the timeout behavior of different IoT devices. In Section 6.4 and Section 6.5, we evaluate the performance of detecting and handling the delay attacks, respectively. Besides, compared to the typical processing time and network latency between an IoT device and its cloud server [39], which is several seconds, the overhead (latency) of MP-Mediator due to the redirection and processing is negligible (hundreds of milliseconds). The communication overhead of MP-Mediator is also evaluated. Details on the overhead are presented in Appendix B due to space limit.

6.1 The Smart Home Testbed

To evaluate the performance of MP-Mediator, we set up a real-world smart home testbed as shown in Figure 6. The testbed is a studio with one resident and a total of 22 IoT devices are deployed in the testbed. Details of the devices are listed in Table 2, including device names, abbreviation labels, wireless protocols, main attributes and link information. The link information indicates how a device connects to MP-Mediator, which is illustrated in Figure 5. Some of the devices, such as Aqara hub (AH) and Blue by ADT security system (SS2), also act as hubs, connecting devices of the same brand that use Zigbee or Z-Wave protocols. For consistency, we let such Zigbee/Z-Wave devices connect to their own hubs, which is the original way. That is, we do not connect such Zigbee/Z-Wave devices directly to the Device Manager. A total of 10 automation rules are installed on two popular cloud integration platforms: IFTTT and SmartThings. Table 3 gives the details of automation rules with the device abbreviations in bold. Some of the rules utilize the user's smartphone as a presence sensor, which is not included in Figure 6 because of the mobility of the user/smartphone, which could be anywhere in the home. Note that a smart plug is used to simulate the control of a water valve due to the restriction on installing a real water valve in the testbed. Implementation details of MP-Mediator are presented in Appendix A due to space limit.

IRB Approval: We have received IRB approval for our experiments. The participant is fully aware of all the installed devices and automation rules. To avoid possible safety issues, the lock is tested only when the participant is at home and is informed in advance about the testing. The data were pre-processed to hide/remove any sensitive and personally identifiable information (PII).

Table 4: Time for MP-Mediator to confirm successful *event* delivery in normal circumstances (unit: second).

Device Name (Abbr.)	Link	Min	Max	Mean	Median
SimpliSafe security system (SS1)	*CDwAPI-VC*	3.426	**4.905**	3.716	3.558
Kwikset Kevo Lock + Kevo Plus (LK)	*CDwAPI-VC*	2.932	**7.326**	5.351	5.355
Ring Video Doorbell 4 (VD)	*CDwAPI-VC*	1.492	**3.637**	2.398	2.347
Eufy indoor camera (CAM)	*CDwAPI-VC*	1.362	**6.144**	3.073	2.860
Aqara hub (with night light) (AH)	*CDwoAPI-VC*	1.328	**4.983**	2.021	1.842
Blue by ADT security system (SS2)	*CDwoAPI-VC*	1.556	**3.793**	1.928	1.839
Yeelight light bulb (L1)	*LD-MPM*	0.493	**1.657**	1.033	0.980
Wemo smart plug (P1)	*LD-MPM*	0.234	**1.629**	0.947	0.965

Table 5: Time for MP-Mediator to confirm successful *command* delivery in normal circumstances (unit: second).

Device Name (Abbr.)	Link	Min	Max	Mean	Median
SimpliSafe security system (SS1)	*CDwAPI-VC*	0.930	**7.809**	1.827	1.496
Kwikset Kevo Lock + Kevo Plus (LK)	*CDwAPI-VC*	0.775	**6.862**	3.090	2.981
Ring Video Doorbell 4 (VD)	*CDwAPI-VC*	1.161	**2.591**	1.697	1.726
Eufy indoor camera (CAM)	*CDwAPI-VC*	1.409	**3.864**	2.315	2.128
Aqara hub (with night light) (AH)	*CDwoAPI-VC*	-0.930	**0.373**	-0.076	-0.012
Blue by ADT security system (SS2)	*CDwoAPI-VC*	0.294	**9.522**	4.827	4.956

6.2 Determining the Timing Thresholds

In order to detect the delay attacks on events and commands, it is necessary to measure how long it takes MP-Mediator to know an event or command has been successfully delivered in normal cases without the delay attacks. We measure the timing for all cloud-connected devices on *CDwAPI-VC* and *CDwoAPI-VC*, as well as some LAN-connected devices on *LD-MPM*. Note that commands for devices on *LD-MPM* cannot be delayed by attackers, and hence we only consider the *event* delay attacks for devices on *LD-MPM*. Recall that the details of event and command flows are given in Figure 5. The related timing measurement can be done automatically in a secure environment when MP-Mediator is deployed in a new IoT system. Specifically, MP-Mediator automatically records the timestamp information (e.g., when sniffing, receiving, and sending events and commands) and then calculates the thresholds accordingly.

6.2.1 Thresholds for Events. For events from *CDwAPI-VC* and *LD-MPM*, we need to measure the time between sniffing an event in LAN and receiving it on the Device Manager. For events from *CDwoAPI-VC*, we need to measure the time between sniffing an

Table 6: Measurement results of device timeout behavior without using MP-Mediator.

Abbr.	Device Name	Link	Long-live Session	Keep-alive Messages			Event Messages		Command Messages	
				Period (s)	Pattern	Timeout (s)	Timeout (s)	Range (s)	Timeout (s)	Range (s)
SS1	SimpliSafe security system*	*CDwAPI-VC*	Yes	55	fixed	30	20	[20,20]	∞	[30,85]
LK	Kwikset Kevo Lock + Kevo Plus	*CDwAPI-VC*	Yes	30	on-idle	180	∞	[180,210]	∞	[180,210]
VD	Ring Video Doorbell 4	*CDwAPI-VC*	Yes	60	on-idle	35	∞	[35,95]	∞	[35,95]
CAM	Eufy indoor camera	*CDwAPI-VC*	No	-	-	-	30	[30,30]	30	[30,30]
AH	Aqara hub (with night light)	*CDwoAPI-VC*	Yes	30	fixed	90	∞	[90,120]	∞	[90,120]
SS2	Blue by ADT security system	*CDwoAPI-VC*	Yes	30	fixed	33	∞	[33,63]	∞	[33,63]
P4	Aqara smart plug	*CDwoAPI-VC*	Yes	30	fixed	90	∞	[90,120]	∞	[90,120]
M1	Aqara motion sensor	*CDwoAPI-VC*	Yes	30	fixed	90	∞	[90,120]	-	-
M3	Blue by ADT motion sensor	*CDwoAPI-VC*	Yes	30	fixed	33	∞	[33,63]	-	-
L1, L2	Yeelight light bulb	*LD-MPM*	Yes	16	on-idle	46	∞	[46,62]	∞	[46,62]
P1	Wemo smart plug*	*LD-MPM*	No	-	-	-	52	[52,52]	15	[15,15]
P2, P3	Kasa smart plug	*LD-MPM*	Yes	200	fixed	30	∞	[30,230]	∞	[30,200]
M2	SmartThings motion sensor*	*LD-MPM*	Yes	31	on-idle	16	∞	[16,47]	-	-
SD	First Alert smoke detector*	*LD-MPM*	Yes	31	on-idle	16	∞	[16,47]	-	-
SR	Aeotec Siren 6*	*LD-MPM*	Yes	31	on-idle	16	∞	[16,47]	-	-
WL	SmartThings leak sensor*	*LD-MPM*	Yes	31	on-idle	16	∞	[16,47]	-	-
B	SmartThings button*	*LD-MPM*	Yes	31	on-idle	16	∞	[16,47]	-	-
L3, L4	Philips Hue light bulb*	*LD-MPM*	Yes	120	fixed	60	∞	[60,180]	21	[21,21]
M4	Philips Hue motion sensor*	*LD-MPM*	Yes	120	fixed	60	∞	[60,180]	-	-

Table 7: Results of detecting *event* delay attacks.

Abbr.	Link	Threshold (s)	Attacker delay (s)	Precision	Recall
SS1	*CDwAPI-VC*	6	20	100%	100%
LK	*CDwAPI-VC*	9	[180,210]	100%	100%
VD	*CDwAPI-VC*	5	[35,95]	100%	100%
CAM	*CDwAPI-VC*	8	30	100%	100%
AH	*CDwoAPI-VC*	6	[90,120]	100%	100%
SS2	*CDwoAPI-VC*	5	[33,63]	96.15%	100%
L1	*LD-MPM*	3	[46,62]	100%	100%
P1	*LD-MPM*	3	52	100%	100%

Table 8: Results of detecting *command* delay attacks.

Abbr.	Link	Threshold (s)	Attacker delay (s)	Precision	Recall
SS1	*CDwAPI-VC*	9	[30,85]	96.15%	100%
LK	*CDwAPI-VC*	8	[180,210]	100%	100%
VD	*CDwAPI-VC*	4	[35,95]	100%	100%
CAM	*CDwAPI-VC*	5	30	100%	100%
AH	*CDwoAPI-VC*	2	[90,120]	100%	100%
SS2	*CDwoAPI-VC*	11	[33,63]	98.04%	100%

event in LAN and receiving a command by the corresponding virtual device (Section 5.2.2). We measure 40 times for each device and the measurement results are presented in Table 4. Minimum, maximum, mean and median are used to capture the distribution of the measured time for each device. On average, it takes no more than 6 seconds for MP-Mediator to confirm a successful event delivery. Especially for devices on *LD-MPM*, it only takes less than 2 seconds due to the low latency of device-to-device communication in LAN. The time range varies for each device. For instance, the SimpliSafe security system has a relatively small variance, which is from 3.426 seconds to 4.905 seconds. Table 4 shows that the Eufy indoor camera has the largest range of time (varies from 1.362 seconds to 6.144 seconds) among all the tested devices, but it is still acceptable. Propagation delay and network conditions vary at different measurements, which is the reason for the time variance. Besides, another possible reason is that different cloud servers may have different implementations of authentication and request scheduling. To choose a suitable threshold for various unpredictable conditions, we set the value to be $\lceil Max \rceil + 1$ for each device, which gives some additional time in case of a bad network condition.

6.2.2 Threshold for Commands. For commands sent via *MPM-IP* by integration platforms, they can successfully reach MP-Mediator due to the heartbeats used to prevent the delay attacks on *MPM-IP*. After that, MP-Mediator forwards commands to target devices via *VC-MPM* and *CDwAPI-VC*. For the above process, we need to measure the time between when MP-Mediator forwards a command to the vendor cloud and the command from the vendor cloud being sniffed in LAN. For commands sent via *VC-IP* by integration platforms, they do not go through MP-Mediator. Thanks to virtual rules and virtual devices, the integration platform issues another command to a virtual device at the same time when issuing a command to a physical device (Section 5.2.2). We need to measure the time between receiving the virtual command on MP-Mediator and sniffing the command in LAN, which is sent to the physical device.

We measure 40 times for each device and Table 5 lists the measurement results. On average, it takes less than 5 seconds for MP-Mediator to confirm a successful command delivery. For *CDwoAPI-VC* devices (AH and SS2), the time values are calculated by using the time of sniffing the real command in LAN minus the time of receiving the virtual command. Note that the real and virtual commands are issued at the same time from the cloud server, and the order of arrival of these two commands is non-deterministic, which depends on cloud servers and network conditions. E.g., for the Aqara hub, sometimes the real command arrives earlier than the virtual command, resulting in negative values in Table 5. For detecting the command delay attacks, we also set the threshold to $\lceil Max \rceil + 1$.

6.3 Device Timeout Behaviour

As studied in a previous work [35], the timeout checking of application layer protocols cannot be bypassed by attackers due to the TLS protection of the application payload, which means that an application layer protocol timeout would occur if an attacker delays an event or command longer than the timeout limit. A delay

Table 9: Results of handling *event* delay attacks.

Abbr.	Link	As event in rules	Handling method	success / total
SS1	*CDwAPI-VC*	None	Send sniffed event to cloud platform	50 / 50
LK	*CDwAPI-VC*	As condition event in R1 and R2	Send sniffed event to cloud platform	50 / 50
VD	*CDwAPI-VC*	None	Send sniffed event to cloud platform	50 / 50
CAM	*CDwAPI-VC*	None	Send sniffed event to cloud platform	50 / 50
AH	*CDwoAPI-VC*	As trigger event in R3	Send R3's command to the target device	50 / 50
SS2	*CDwoAPI-VC*	As condition event in R9	Notify user about the event delay attack	50 / 50
L1	*LD-MPM*	None	Send sniffed event to cloud platform	50 / 50
P1	*LD-MPM*	None	Send sniffed event to cloud platform	50 / 50

attacker may want to delay an event or command as long as possible, without causing the time out and termination of a connection, which maximizes the consequences of the delay attacks.

Table 6 lists the measurement results of device timeout behaviors without using MP-Mediator. For devices that are marked with asterisks, their timeout behaviors have already been evaluated in the previous work [35]. In our work, we measure the timeout behaviors of 10 new devices by using the same method as in [35]. The Eufy indoor camera and the Wemo smart plug establish on-demand sessions with their cloud servers only when sending events and receiving commands. All other devices maintain long-live TCP and TLS sessions with their cloud servers following some patterns, which can be divided into two categories: *fixed* and *on-idle*. A fixed pattern means that the keep-alive messages are exchanged at a fixed period of time no matter whether the session is idle or not, and on-idle means that the pattern is non-periodically due to occasional activities. An "∞" symbol indicates the timeout for the device is only triggered by keep-alive messages and the device does not have a specific timeout for the event and command messages. The "Period" and "Timeout" parameters of keep-alive messages indicate the time an attacker can delay a session. For example, if an attacker delays a SimpliSafe command right before a keep-alive message is sent, the maximum time the attacker can delay is 30 seconds. If right after a keep-alive message, the attacker can delay up to: Period + Timeout (55+30) = 85 seconds.

6.4 Performance of Detecting Delay Attacks

We evaluate the performance of detecting the delay attacks on events and commands in Section 6.4.1 and Section 6.4.2, respectively. Similar to Section 6.2, we only evaluate the detection of the *event* delay attacks for devices on *LD-MPM*. We simulate WAN delay attacks by launching the delay attacks on the same links in LAN.

6.4.1 Detecting Event Delay Attacks. We trigger 100 events for each device, and 50 events are sent without delay attacks. We simulate an attacker to delay the other 50 events according to the longest time that each device event can be delayed. Table 7 presents the detection results for both normal (Negative) and delayed events (Positive). The thresholds used for detection are calculated by $\lceil Max \rceil + 1$ as mentioned in Section 6.2. As shown in Table 7, MP-Mediator achieves a precision of over 96% and a recall of 100% in detecting event delay attacks. Specifically, it only mistakenly takes two events of the ADT security system as being delayed, which is due to a very long network delay. Besides, MP-Mediator can detect all delayed events based on the chosen thresholds within a short time, which significantly reduces the amount of time an attacker can delay.

Table 10: Results of handling *command* delay attacks.

Abbr.	Link	Time to re-establish (s)	Re-sending Cmd. delay (s)	Attacker delay (s)	success / total
SS1	*CDwAPI-VC*	10	19 (= 9 + 10)	[30,85]	50 / 50
LK	*CDwAPI-VC*	40	48 (= 8 + 40)	[180,210]	50 / 50
VD	*CDwAPI-VC*	25	29 (= 4 + 25)	[35,95]	50 / 50
CAM	*CDwAPI-VC*	On demand	5.2 (= 5 + 0.2)	30	50 / 50
AH	*CDwoAPI-VC*	4	6 (= 2 + 4)	[90,120]	50 / 50
SS2	*CDwoAPI-VC*	15	26 (= 11 + 15)	[33,63]	50 / 50

6.4.2 Detecting Command Delay Attacks. Similarly, we issue 100 commands for each device. Fifty of them are sent without delay attacks. We simulate an attacker to delay the other 50 commands according to the device's longest command delay in Table 6. Results are presented in Table 8, which indicates that MP-Mediator also has a very high precision of over 96% and a recall of 100% in detecting command delay attacks. It takes only a few seconds to detect command delay attacks for most of the devices. Our detection mechanism significantly reduces the adversary effect of the command delay attacks, which can cause a long delay (tens of seconds to several minutes) on important commands.

6.5 Performance of Handling Delay Attacks

In Section 6.4, we show that MP-Mediator can accurately detect the delay attacks. Handling the attacks is also important. We evaluate the handling of the delay attacks on events and commands in Section 6.5.1 and Section 6.5.2, respectively. Note that some handling approaches are based on the urgency of the affected rule. If a rule is not urgent, then the potential damage caused by the attack is small. We want to focus on urgent issues. Hence, for the experiments presented in this subsection, all the rules in Table 3 are considered urgent and the handling methods are evaluated.

6.5.1 Handling Event Delay Attacks. Once an event delay attack is detected, MP-Mediator handles it according to the automation rules associated with the sniffed event. If the sniffed event is a trigger event for some rule, MP-Mediator directly sends the expected command to the target device to fulfill the execution of the automation rule. If the sniffed event is a condition event for some rule, MP-Mediator sends the sniffed event to the cloud platform to update the device's status in a timely manner. Table 9 shows that MP-Mediator is very effective in handling event delay attacks. MP-Mediator successfully handles all 50 event delay attacks for each device, and takes action to keep the smart home secure and device status up-to-date. Please note, even if MP-Mediator sends a duplicate event to the cloud platform due to an unintentional delay in the event message, it will not alter the state of the corresponding device. Therefore, it does not cause any operational issues for the home automation system.

6.5.2 Handling Command Delay Attacks. Handling command delay attacks is more crucial since commands are actions that should be taken to keep the smart home ecosystem safe and secure. As mentioned in Section 5.2.1, when a command delay attack happens on *CDwAPI-VC* and *VC-MPM*, we cannot determine the exact link. Thus, when handling command delay attacks on *CDwAPI-VC* and *VC-MPM*, we combine the solutions for both attack scenarios. Specifically, MP-Mediator issues the command on a smartphone using a cellular network to ensure the command can be successfully delivered to the vendor cloud server, and then uses the VPN-based method to send the command to the target device.

Upon detecting a command delay attack, MP-Mediator immediately terminates the device-side TCP connection at the Device Traffic Monitor by shutting down the socket and then starts a VPN service to tunnel the new connection. The IoT device thinks the cloud server terminates the previous connection, and it tries to establish a new connection with the cloud server. Evaluation results of handling command delay attacks are shown in Table 10. Note that the Eufy indoor camera does not maintain long-live sessions. The connection with its cloud server is on-demand and can be established instantly (about 0.2 seconds). The total time of re-sending a command is also measured and listed in Table 10, which is calculated by adding the threshold and the time for re-establishment. The results show that terminating a previous connection and then establishing a new one takes some time (which is the last part in column 4 of Table 10). The on-demand case has no previous connection to terminate and can immediately establish a new connection, so it is very short (0.2 seconds). All 50 commands of each device can be successfully re-sent via the VPN tunnel before the attacker finishes the delay. The attacker can delay some of the commands (e.g., a command for AH) up to 120 seconds, but it only takes 6 seconds for MP-Mediator to detect and re-send them, which greatly reduces the delay of the commands and protects the smart home.

7 RELATED WORK

The stealthy delay attacks on events and commands in smart home environments are newly discovered. To the best of our knowledge, there is no prior work on detecting and handling the new attacks. In this section, we discuss and clarify why existing solutions are not suitable for protecting smart homes against the new delay attacks. Jamming attacks are the most similar to the stealthy delay attacks, which we discuss in Appendix C due to space limit.

Exploration of misordered messages. Some prior works [27, 39] studied the impact of misordered events/commands and proposed some solutions to mitigate them. A secure service named Omega is proposed in [27] offering guarantees regarding the ordering of events. However, the secure service relies on special hardware security modules such as Intel SGX [28] and requires the devices to install edge computing client software, which includes asymmetric encryption operations. Most smart home IoT devices do not have special hardware security modules and do not have the resources to run asymmetric encryption. These limitations make it impractical to protect billions of existing IoT devices that have been deployed but do not have the hardware security modules and/or computation resources. Another recent paper [39] only measures the latency of cloud-based IoT services and does not discuss any mitigation methods at all. Our work is the first to discuss the detection and handling of the new delay attacks [35].

Policy-based approaches. Some existing works [24, 31] propose run-time policy enforcement systems to detect unsafe and insecure device states using security policies specified by users/experts or learned from the environmental context. These policy-based methods may be able to detect some of the delay attacks if the delayed devices cause violations of one of the pre-defined policies. However, policy-based methods have the following issues:

1) It is very difficult (if possible) to ensure a set of pre-defined policies is *complete*, i.e., the pre-defined policies cover all possible scenarios/attacks. For example, delaying the event of a presence sensor becoming not-present (when the user is leaving home) does not violate any policies listed in [24], because activities are considered legitimate when the user is at home (the system mistakenly believes that). However, suppose this presence sensor's event is used as the trigger to automatically lock the smart lock/door, delaying it causes the locking action to be delayed accordingly after the user leaves home, which increases the risk of burglary.

2) It is difficult to ensure that a set of pre-defined policies is *accurate* for different homes, which may have different IoT devices installed in different rooms/locations. Furthermore, unexpected and complex situations can arise in a smart home environment that the pre-defined policies cannot correctly handle. E.g., for the policy defined in [24]: "*The windows must not be open when the heater is on*", it prohibits windows from opening even if the heater causes a fire, which may further cause serious harm to residents due to smoke buildup and/or carbon monoxide poisoning.

3) Moreover, even if the policies are accurate and complete (which is very difficult if not impossible), the policy enforcement systems can only handle the delay attacks after they happen, but cannot detect the delay attacks when they initially occur. Without timely detection, severe consequences could happen to a smart home. For instance, considering a policy "*Turn on sprinkler when the smoke detector detects fire*", if a delay attack delays the event of the smoke detector, the policy cannot be enforced since the policy enforcement system is not aware of this critical event, which may cause fire undetected in time and threaten the safety of the residents. In such situations, every second matters. On the other hand, MP-Mediator can detect the delayed event of the smoke detector in a timely manner when it happens and immediately take action, e.g., turning on the sprinkler and alerting the residents.

Anomaly detection systems. Recent works [20, 21, 36, 54] utilize physical context among IoT devices to detect anomalies of intercepted/masked events and commands, which have similar consequences as the new delay attacks studied in this paper. However, these anomaly detection methods are not suitable for detecting the new delay attacks. In general, physical context-based anomaly detection methods require correct and up-to-date device states in order to reflect the ever-changing real-world physical environment. To this end, they rely on various sensors to collect real-time sensor events from the physical environment, but the sensors themselves are vulnerable to the delay attacks (e.g., sensor events being delayed), causing an inconsistency between the real-world environment and the context generated by the out-of-date device states.

Peeves [20] and Haunted House [21] can only detect anomalies of actuator (but not sensor) events that cause changes in the physical

environment, while MP-Mediator can detect the delay attacks on both events and commands of both sensors and actuators. Besides, both Peeves and Haunted House require a high sampling rate on numeric sensor data to capture changes in the physical environment, which requires specialized hardware and causes a large overhead. Compared to HAWatcher [36], MP-Mediator does not need to mine or rely on correlations as HAWatcher does. Besides, HAWatcher has a 60-second timeout threshold for anomaly detection and it cannot detect anomalies if the delay is shorter than 60 seconds. The timeout thresholds for MP-Mediator are much shorter than 60 seconds and delay attacks can be quickly detected. Aegis [54] focuses on differentiating benign and specific types of malicious activities/behaviors based on the contextual model, which is essentially a group of device states updated according to the received sensor/device events. As mentioned, the context model itself is vulnerable to the event delay attacks.

8 DISCUSSION

VPN-based method. The use of the VPN-based function significantly enhances the capability of MP-Mediator in handling delayed commands. There are out-of-box configuration templates and scripts for non-technical users that provide a one-click setup of the VPN-related functions. The additional cost of resources is negligible since the VPN-based function is only used when a command delay attack is detected. In normal situations, there is no traffic transmitted through the VPN tunnel. This makes it possible to deploy the VPN-based function on many free-tier public services, such as Google Cloud Free Program [5]. Optionally, users can also disable the function temporarily in low-risk periods, during which they can still receive alerts when the command delay attacks are detected.

Timeout threshold poisoning attack. As mentioned in Section 6.2, the timeout thresholds used for detecting delay attacks need to be measured in a secure environment. In the worst case, an attacker may conduct delay attacks during the threshold measuring stage, which poison the timing information and result in inaccurate thresholds. To mitigate this issue, MP-Mediator can perform random sampling on the data before calculating timeout thresholds.

Adaptive attackers. The detection timeout threshold is set to be $\lceil Max \rceil + 1$ for each device (Section 6.2). Although an adaptive attacker who knows the threshold used for the detection may conduct delay attacks up to the defined threshold without being detected, the "Threshold" and "Attacker delay" columns in Table 7 and Table 8 show that: the timeout-based method significantly limits the time an adaptive attacker can delay and reduces the consequences.

Alternative threshold choosing and fine-tuning. While a diverse range of IoT devices requires timeout threshold measurements, users can conveniently share these values via online forums and communities. This allows others with the same device model to apply the values directly without measuring again. Considering the variety of network conditions in different smart homes, users can adjust the thresholds according to their preferences, balancing the trade-off between security and false alarms based on the shared values.

UDP protocol. In this work, we demonstrate the design of MP-Mediator with IoT devices that utilize TCP connections because TCP is still the dominating transport layer protocol being used by IoT devices [41]. Even for devices using UDP, MP-Mediator still performs well on monitoring devices' traffic by using the metadata in their application layer protocols such as CoAP [2] and QUIC [13].

9 CONCLUSION

In this work, we presented MP-Mediator, a module that can effectively detect and handle the stealthy delay attacks, which can be launched on most smart home/office IoT systems. This is the first study that addresses the new family of attacks, which could cause severe consequences (e.g., CO poisoning, fire, and burglary) in smart home/office. We proposed novel virtual-device and virtual-rule based approaches, as well as other effective approaches to detect and handle the delay attacks. For the command delay attack, we presented a VPN-based method that immediately sends a delayed command via a secret tunnel right after detecting the attack. We implemented MP-Mediator in a real-world smart home testbed that connects twenty-two IoT devices and two popular integration platforms. We evaluated the performance of MP-Mediator and showed that MP-Mediator can quickly detect and effectively handle the delay attacks on both IoT events and commands for various scenarios.

ACKNOWLEDGMENTS

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

REFERENCES

[1] 2023. Amazon Alexa. https://developer.amazon.com/.
[2] 2023. Constrained Application Protocol. https://en.wikipedia.org/wiki/Constrained_Application_Protocol.
[3] 2023. Eclipse Mosquitto - An open source MQTT broker. https://mosquitto.org/.
[4] 2023. GoControl HUSBZB-1 USB Stick. https://www.gocontrol.com/detail.php?productId=6.
[5] 2023. Google Cloud Free Program. https://cloud.google.com/free/docs/gcp-free-tier/#compute.
[6] 2023. Home Assistant – Open source home automation that puts local control and privacy first. https://www.home-assistant.io/.
[7] 2023. Homebridge – Bringing HomeKit support where there is none. https://homebridge.io/.
[8] 2023. HomKit. https://developer.apple.com/homekit/.
[9] 2023. IFTTT - Every thing works better together. https://ifttt.com/.
[10] 2023. ioBroker – Automate your life. https://www.iobroker.net/.
[11] 2023. openHAB – an open-source platform for empowering home automa-tion. https://www.openhab.org/.
[12] 2023. OpenWrt. https://openwrt.org/.
[13] 2023. QUIC. https://en.wikipedia.org/wiki/QUIC.
[14] 2023. SmartThings. https://www.smartthings.com/.
[15] 2023. SONOFF ZigBee 3.0 - a universal Zigbee USB stick.
[16] 2023. Z-Wave JS UI - Fully configurable Zwave to MQTT Gateway and Control Panel. https://github.com/zwave-js/zwave-js-ui.
[17] 2023. Zigbee2MQTT - Zigbee to MQTT bridge, get rid of your proprietary Zigbee bridges. https://www.zigbee2mqtt.io/.
[18] Abbas Acar, Hossein Fereidooni, Tigist Abera, Amit Kumar Sikder, Markus Miettinen, Hidayet Aksu, Mauro Conti, Ahmad-Reza Sadeghi, and Selcuk Uluagac. 2020. Peek-a-boo: I see your smart home activities, even encrypted!. In *Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks*. 207–218.
[19] Emrah Bayraktaroglu, Christopher King, Xin Liu, Guevara Noubir, Rajmohan Rajaraman, and Bishal Thapa. 2013. Performance of IEEE 802.11 under jamming. *Mobile Networks and Applications* 18, 5 (2013), 678–696.
[20] Simon Birnbach and Simon Eberz. 2019. Peeves: Physical event verification in smart homes. (2019).
[21] Simon Birnbach, Simon Eberz, and Ivan Martinovic. 2022. Haunted House: Physical Smart Home Event Verification in the Presence of Compromised Sensors. *ACM Transactions on Internet of Things* 3, 3 (2022), 1–28.
[22] Ioannis Broustis, Konstantinos Pelechrinis, Dimitris Syrivelis, Srikanth V Krishnamurthy, and Leandros Tassiulas. 2009. FIJI: Fighting implicit jamming in 802.11

WLANs. In *International Conference on Security and Privacy in Communication Systems*. Springer, 21–40.
[23] Z Berkay Celik, Patrick McDaniel, and Gang Tan. 2018. Soteria: Automated {IoT} Safety and Security Analysis. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. 147–158.
[24] Z Berkay Celik, Gang Tan, and Patrick D McDaniel. 2019. IoTGuard: Dynamic Enforcement of Security and Safety Policy in Commodity IoT.. In *NDSS*.
[25] Haotian Chi, Chenglong Fu, Qiang Zeng, and Xiaojiang Du. 2022. Delay Wreaks Havoc on Your Smart Home: Delay-based: Automation Interference Attacks. In *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 1575–1575.
[26] Haotian Chi, Qiang Zeng, Xiaojiang Du, and Jiaping Yu. 2020. Cross-app interference threats in smart homes: Categorization, detection and handling. In *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 411–423.
[27] Cláudio Correia, Miguel Correia, and Luís Rodrigues. 2020. Omega: a secure event ordering service for the edge. In *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 489–501.
[28] Victor Costan and Srinivas Devadas. 2016. Intel SGX explained. *Cryptology ePrint Archive* (2016).
[29] Asish Kumar Dalai and Sanjay Kumar Jena. 2017. Wdtf: A technique for wireless device type fingerprinting. *Wireless Personal Communications* 97, 2 (2017), 1911–1928.
[30] Christian Dietz, Raphael Labaca Castro, Jessica Steinberger, Cezary Wilczak, Marcel Antzek, Anna Sperotto, and Aiko Pras. 2018. IoT-botnet detection and isolation by access routers. In *2018 9th International Conference on the Network of the Future (NOF)*. IEEE, 88–95.
[31] Wenbo Ding, Hongxin Hu, and Long Cheng. 2021. IOTSAFE: Enforcing safety and security policy with real IoT physical interaction discovery. In *the 28th Network and Distributed System Security Symposium (NDSS 2021)*.
[32] Alaba Ayotunde Fadele, Mazliza Othman, Ibrahim Abaker Targio Hashem, Ibrar Yaqoob, Muhammad Imran, and Muhammad Shoaib. 2019. A novel countermeasure technique for reactive jamming attack in internet of things. *Multimedia Tools and Applications* 78, 21 (2019), 29899–29920.
[33] Nick Farina. [n. d.]. homebridge-dummy. https://www.npmjs.com/package/homebridge-dummy. (Accessed on 10/10/2022).
[34] Earlence Fernandes, Amir Rahmati, Kevin Eykholt, and Atul Prakash. 2017. Internet of things security research: A rehash of old ideas or new intellectual challenges? *IEEE Security & Privacy* 15, 4 (2017), 79–84.
[35] Chenglong Fu, Qiang Zeng, Haotian Chi, Xiaojiang Du, and Siva Likitha Valluru. 2022. Iot phantom-delay attacks: Demystifying and exploiting iot timeout behaviors. In *2022 52st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE.
[36] Chenglong Fu, Qiang Zeng, and Xiaojiang Du. 2021. HAWatcher: Semantics-Aware Anomaly Detection for Appified Smart Homes. In *30th USENIX Security Symposium (USENIX Security 21)*. 4223–4240.
[37] P Ganeshkumar, KP Vijayakumar, and M Anandaraj. 2016. A novel jammer detection framework for cluster-based wireless sensor networks. *EURASIP Journal on Wireless Communications and Networking* 2016, 1 (2016), 1–25.
[38] Gigamon. 2016. Understanding network taps – the first step to visibility. https://www.gigamon.com/resources/resource-library/white-paper/understanding-network-taps-first-step-to-visibility.html.
[39] Furkan Goksel, Muslum Ozgur Ozmen, Michael Reeves, Basavesh Shivakumar, and Z Berkay Celik. 2021. On the safety implications of misordered events and commands in IoT systems. In *2021 IEEE Security and Privacy Workshops (SPW)*. IEEE, 235–241.
[40] Tianbo Gu, Zheng Fang, Allaukik Abhishek, and Prasant Mohapatra. 2020. IoTSpy: Uncovering Human Privacy Leakage in IoT Networks via Mining Wireless Context. In *2020 IEEE 31st Annual International Symposium on Personal, Indoor and Mobile Radio Communications*. IEEE, 1–7.
[41] Danny Yuxing Huang, Noah Apthorpe, Frank Li, Gunes Acar, and Nick Feamster. 2020. Iot inspector: Crowdsourcing labeled network traffic from smart home devices at scale. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 4, 2 (2020), 1–21.
[42] Yeonseon Jeong, Hyunghoon Kim, and Hyo Jin Jo. 2022. ASD: ARP Spoofing Detector Using OpenWrt. *Security and Communication Networks* 2022 (2022).
[43] Yunhan Jack Jia, Qi Alfred Chen, Shiqi Wang, Amir Rahmati, Earlence Fernandes, Zhuoqing Morley Mao, Atul Prakash, and SJ Unviersity. 2017. ContexIoT: Towards Providing Contextual Integrity to Appified IoT Platforms.. In *NDSS*, Vol. 2. San Diego, 2–2.
[44] Mingyan Li, Iordanis Koutsopoulos, and Radha Poovendran. 2007. Optimal jamming attacks and network defense policies in wireless sensor networks. In *IEEE INFOCOM 2007-26th IEEE International Conference on Computer Communications*. IEEE, 1307–1315.
[45] Yuan Luo, Long Cheng, Hongxin Hu, Guojun Peng, and Danfeng Yao. 2020. Context-Rich Privacy Leakage Analysis Through Inferring Apps in Smart Home IoT. *IEEE Internet of Things Journal* 8, 4 (2020), 2736–2750.
[46] Sergey A Marchenkov, Dmitry G Korzun, Anton I Shabaev, and Anatoly V Voronin. 2017. On applicability of wireless routers to deployment of smart spaces in Internet of Things environments. In *2017 9th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, Vol. 2. IEEE, 1000–1005.
[47] Christoph Mayer. [n. d.]. XArp. http://www.xarp.net/. (Accessed on 09/20/2022).
[48] Markus Miettinen, Samuel Marchal, Ibbad Hafeez, N Asokan, Ahmad-Reza Sadeghi, and Sasu Tarkoma. 2017. Iot sentinel: Automated device-type identification for security enforcement in iot. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2177–2184.
[49] Danh Nguyen, Cem Sahin, Boris Shishkin, Nagarajan Kandasamy, and Kapil R Dandekar. 2014. A real-time and protocol-aware reactive jamming framework built on software-defined radios. In *Proceedings of the 2014 ACM workshop on Software radio implementation forum*. 15–22.
[50] Andrea Di Pasquale. 2016. ArpON. https://arpon.sourceforge.io/. (Accessed on 09/20/2022).
[51] Konstantinos Pelechrinis, Marios Iliofotou, and Srikanth V Krishnamurthy. 2010. Denial of service attacks in wireless networks: The case of jammers. *IEEE Communications surveys & tutorials* 13, 2 (2010), 245–257.
[52] Alan Reed. 2017. ARP-Defense. https://github.com/aarreedd/ARP-Defense. (Accessed on 09/20/2022).
[53] Mustafizur R Shahid, Gregory Blanc, Zonghua Zhang, and Hervé Debar. 2018. IoT devices recognition through network traffic analysis. In *2018 IEEE international conference on big data (big data)*. IEEE, 5187–5192.
[54] Amit Kumar Sikder, Leonardo Babun, Hidayet Aksu, and A Selcuk Uluagac. 2019. Aegis: A context-aware security framework for smart home systems. In *Proceedings of the 35th Annual Computer Security Applications Conference*. 28–41.
[55] Statista. [n. d.]. Smart home device penetration in the U.S. 2021. https://www.statista.com/statistics/1247351/smart-home-device-us-household-penetration/. (Accessed on 12/01/2022).
[56] Alanoud Subahi and George Theodorakopoulos. 2019. Detecting IoT user behavior and sensitive information in encrypted IoT-app traffic. *Sensors* 19, 21 (2019), 4777.
[57] Rahmadi Trimananda, Janus Varmarken, Athina Markopoulou, and Brian Demsky. 2020. Packet-level signatures for smart home devices. In *Network and Distributed Systems Security (NDSS) Symposium*, Vol. 2020.
[58] VERACODE. 2021. ARP Spoofing. https://www.veracode.com/security/arp-spoofing.
[59] Sean Whalen. 2001. An introduction to arp spoofing. *Node99 [Online Document]* (2001).
[60] Kai Yang, Qiang Li, and Limin Sun. 2019. Towards automatic fingerprinting of IoT devices in the cyberspace. *Computer Networks* 148 (2019), 318–327.
[61] Lingjing Yu, Bo Luo, Jun Ma, Zhaoyu Zhou, and Qingyun Liu. 2020. You Are What You Broadcast: Identification of Mobile and {IoT} Devices from (Public){WiFi}. In *29th USENIX Security Symposium (USENIX Security 20)*. 55–72.
[62] Wei Zhang, Yan Meng, Yugeng Liu, Xiaokuan Zhang, Yinqian Zhang, and Haojin Zhu. 2018. Homonit: Monitoring smart home apps from encrypted traffic. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 1074–1088.

A IMPLEMENTATION OF MP-MEDIATOR

The Main Body of MP-Mediator. In recent years, a lot of open-source smart home hubs and platforms have emerged to provide users with more choices for building their own smart home automation systems, such as openHAB [11], Home Assistant [6], Homebridge [7], ioBroker [10] and so on. These emerging hubs and platforms can be deployed on a local PC or laptop to realize local control of IoT devices, and also facilitate the integration of various IoT devices that use different communication technologies (e.g., Bluetooth, Zigbee, Z-Wave, and Wi-Fi). Since Homebridge has a very active community that continuously contributes useful integration (i.e., Homebridge plugins), we choose to use it in our implementation for the main body of MP-Mediator. By now, over 2,000 Homebridge plugins are available to the public, supporting thousands of different IoT devices [7]. The virtual devices mentioned in Section 5.2 and Section 5.3 can be created by using one of the Homebridge plugins [33]. Homebridge was originally developed for HomeKit [8], which allows users to use plugins to integrate with smart devices that do not natively support HomeKit. Later,

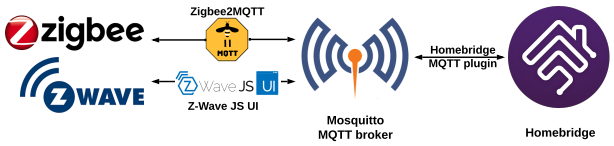


Figure 7: A MQTT broker is used to connect Zigbee and Z-Wave devices to Homebridge.

Homebridge has been greatly extended to work with several cloud integration platforms, such as SmartThings [14], Amazon Alexa [1], and IFTTT [9]. In our implementation, MP-Mediator runs on a laptop with Ubuntu 20.04 LTS, and a two-core Intel(R) Core(TM) i5-5200U CPU. In general, MP-Mediator can run on any computing device (e.g., a Raspberry Pi) as long as Homebridge is supported. Besides, we use a SONOFF Zigbee 3.0 USB Dongle Plus [15] and a GoControl HUSBZB-1 USB Stick [4] to extend Zigbee and Z-Wave capabilities for MP-Mediator, respectively.

Connecting Devices on *CDwAPI-VC*. For IoT devices that only have cloud APIs, they cannot be directly connected to the Device Manager without going through their vendor cloud servers. Fortunately, there are plenty of Homebridge plugins available for these devices. During the configuration stage of a cloud API-based plugin, the user enters the username and password of her account that is associated with the device, in order to pass the authentication process enforced by the cloud server. For some devices, two-factor authentication is required. As shown in Figure 5, a device on *CDwAPI-VC* sends an event to its vendor cloud and its state is updated on the vendor cloud accordingly. In order to keep the device status up-to-date on MP-Mediator, the device plugin utilizes cloud APIs to periodically poll the device status from the vendor cloud server. If MP-Mediator needs to send a command to a device on *CDwAPI-VC*, the Device Manager first sends a request to the vendor cloud using its cloud APIs. Then the cloud server processes the request and issues the command to the target device if it finds the request to be legitimate.

Connecting Devices on *CDwoAPI-VC*. These devices cannot be connected to the Device Manager (or any other device), because they do not have local or cloud APIs. To address this issue, we propose to use virtual devices and virtual rules. For each device connected on *CDwoAPI-VC*, a virtual device is created on MP-Mediator and a virtual rule is also generated on the integration platform to establish a correspondence between the physical device and the virtual device. A state change of the physical device is reflected by the virtual device on MP-Mediator using a virtual rule, so that MP-Mediator is aware of the occurrence of device events on *CDwoAPI-VC*. More details about the usage of virtual devices and virtual rules are presented in Section 5.2.

Connecting Wi-Fi Devices with Local APIs on *LD-MPM*. More and more Wi-Fi devices are built with local APIs to enable direct device-to-device communications with other devices in the same home local area network, which protects the user's privacy by conducting inter-device interactions using only local traffic, and also improves a user experience by reducing the latency and providing basic system operability when Internet access is unavailable. Though some of the IoT vendors do not release official local APIs, the active community manages to dig out many local APIs by reverse engineering and then shares them with the public, which facilitates the development of Homebridge plugins to integrate IoT devices based on local APIs.

Usually, there is more than one local API-based plugin available for a device on *LD-MPM*. We choose to use plugins that have been verified since verified plugins are reviewed and endorsed by the Homebridge project team, making them more reliable and trustworthy.

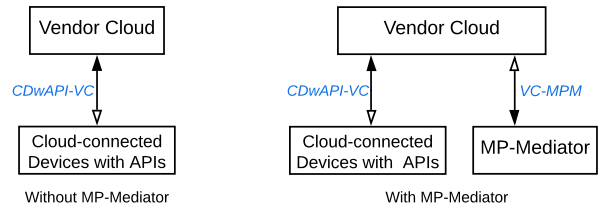
Connecting Zigbee and Z-Wave Devices on *LD-MPM*. As illustrated in Figure 7, instead of directly connecting Zigbee/Z-Wave devices to Homebridge, we utilize a Message Queuing Telemetry Transport (MQTT) broker as a relay. The MQTT broker is created using the open-source project Mosquitto [3] and is hosted on the same computer as MP-Mediator. Besides, we adopt two gateway applications, Zigbee2MQTT [17] and Z-Wave JS UI [16] (formerly ZWavejs2MQTT), which convert Zigbee and Z-Wave protocols to MQTT, respectively. After conversion, the gateway applications publish device event messages to the MQTT broker. In addition, a Homebridge MQTT plugin is used on MP-Mediator for receiving desired MQTT messages from the MQTT broker by subscribing to some specific event topics and MP-Mediator updates the device status accordingly. On the other hand, the Homebridge MQTT plugin publishes control commands to the MQTT broker if there is any command that needs to be sent to target devices. The gateway applications receive the desired commands by subscribing to some command topics and then convert the messages from MQTT to Zigbee/Z-Wave protocols. Finally, the target devices receive the commands and execute them.

Integration with SmartThings. For the IoT devices that have been connected with MP-Mediator, we connect them to the SmartThings platform using *SmartThings Schema* provided by SmartThings. SmartThings provides a list of pre-defined `device handler types` and each device using *SmartThings Schema* must be associated with one of them. A `device handler type` is essentially the profile of a device, indicating what capabilities the device has. For example, a device with the handler type `c2c-contact` implies three capabilities: `Contact Sensor`, `Battery`, `Temperature Measurement`.

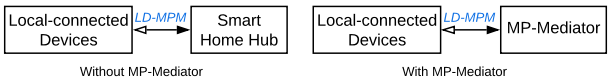
After a successful authentication and authorization process, SmartThings sends `discoveryRequest` to MP-Mediator to request a list of existing devices on MP-Mediator. We need to abstract each device to one of the `device handler types` based on its capabilities and then wrap up the information of all devices in the required format as `discoveryResponse`, which is sent back to SmartThings. Every time the Device Manager receives a device event from the connected device, the Platform Interface forwards the event to SmartThings to update the device status on the SmartThings platform. When the SmartThings platform issues a command to the target device, it sends `commandRequest` to the Platform Interface, which processes the request. Then the Device Manager issues the command to the indicated device.

Table 11: Overhead introduced by MP-Mediator.

Unit: millisecond	Min	Max	Mean	Median	Upper Quartile
Event	250	272	264	264	266
Command	45	634	155	95	199



(a) Cloud-connected devices with cloud APIs.



(b) Local-connected devices with local APIs.

Figure 8: The difference in connection with and without MP-Mediator. Solid arrows present event flows and hollow arrows present command flows.

Integration with IFTTT. Different from the SmartThings platform, IFTTT is an automation tool that connects different services as triggers and actions, instead of maintaining or managing devices on the cloud server. Therefore, we are not required to provide the connected devices to IFTTT. We only need to handle requests that are sent to the Platform Interface from IFTTT when it wants to access the devices that are connected to MP-Mediator. Specifically, when a trigger event occurs, instead of directly pushing the event to IFTTT, the Platform Interface notifies IFTTT of the unique ID of the device that generates the event, indicating the change of the device status. Shortly after receiving the notification, IFTTT sends a request to confirm the change of the device status. If confirmed, according to the defined automation rules, IFTTT issues a command request to the Platform Interface. The Device Manager sends the command to the target device after the command request is successfully processed.

B OVERHEAD OF MP-MEDIATOR

According to [39], the typical processing time and network latency between an IoT device and its cloud server are several seconds. For MP-Mediator, we measure the processing overhead and communication overhead, and present the results in the following. Note that MP-Mediator does not affect unrelated network traffic other than devices that are connected to it.

Processing Overhead. MP-Mediator sits in-between IoT platforms and IoT devices connected via *LD-MPM* or *CDwAPI-VC*. First, MP-Mediator needs to process event messages from IoT devices (or command messages from IoT platforms to devices), and then it forwards the processed events and commands to their destinations. Here we evaluate the overhead introduced by MP-Mediator for processing events and commands. Specifically, we evaluate: 1) the time between MP-Mediator receiving an event and forwarding the event to the IoT platform; 2) the time between MP-Mediator receiving a command and forwarding the command to the target device. We trigger events from an IoT device 50 times, and issue commands to the IoT device 50 times from an IoT platform.

The measuring results are given in Table 11. The time of processing events is relatively stable (around 260 milliseconds), which fluctuates within a small range. For processing commands, the overhead varies from 45 to 600+ milliseconds, with a mean value of 155 milliseconds. Compared to the several seconds of processing time and network latency between IoT devices and servers [39], the overhead of the MP-Mediator processing is negligible.

Table 12: Communication overhead on *VC-MPM*.

Unit: second	Min	Max	Mean	Median	Upper Quartile
Event	0.502	1.875	1.112	1.086	1.346
Command	0.527	2.395	1.039	0.958	1.151

Table 13: Comparison of overhead on *LD-MPM*.

Unit: second	Control Center	Min	Max	Mean	Median	Upper Quartile
Event	Smart Home Hub	0.037	0.151	0.081	0.077	0.101
	MP-Mediator	0.272	1.389	0.981	1.032	1.152
Command	Smart Home Hub	0.055	0.222	0.107	0.091	0.137
	MP-Mediator	0.048	0.249	0.151	0.156	0.178

Communication Overhead. The Device Traffic Monitor redirects network flows between cloud-connected devices and their vendor cloud servers, which adds a small overhead (less than 200 milliseconds). Besides, MP-Mediator requires to obtain device events either from cloud servers (i.e., devices with cloud APIs) or from devices directly (i.e., devices with local APIs), which introduces communication overhead compared to the original connection without using MP-Mediator. The difference in connection is shown in Figure 8. We do not measure communication overhead for the devices without APIs since their events do not go through MP-Mediator.

For cloud-connected devices in Figure 8(a), an extra communication link (i.e., *VC-MPM*) is added between MP-Mediator and the vendor cloud in order for MP-Mediator to poll device status from and send commands to the vendor cloud using cloud APIs. During the measurement, we poll the device status 50 times and send commands to a device 50 times. The measured communication overhead is reported in Table 12. The average communication overhead of event is slightly greater than 1 second, with the minimum of only half a second. The communication overhead on command is also small with the minimum being 0.527 seconds and most of them are around 1 second (upper quartile is 1.151 seconds). It is worth noting that the original connection (i.e., without MP-Mediator) only connects devices to their vendor cloud and some devices (e.g., SimpliSafe security system) cannot be integrated with third-party platforms such as the SmartThings platform. MP-Mediator provides the ability to connect such devices to other third-party platforms, with acceptable communication overhead on *VC-MPM*.

For local-connected devices shown in Figure 8(b), using MP-Mediator does not add any extra communication link. Therefore,

we compare the communication overhead on *LD-MPM* for the two scenarios and the results are reported in Table 13. For Event, MP-Mediator only adds about 1 second overhead, which will not affect the normal operation of smart homes (e.g., automation rules). For command, the communication overhead is similar when using a smart home hub and the MP-Mediator, and it is no more than 0.25 seconds.

C JAMMING ATTACKS

Jamming attacks [19, 22, 44, 49, 51] have been widely studied, including constant, deceptive, random, and reactive jamming, where reactive jamming is the most challenging attack and also more dangerous than other jamming attacks in terms of performance [37]. The jammer in a reactive jamming attack remains silent at ordinary times and sends jamming signals only when it detects a device is sending packets to other devices, causing a collision to destroy packet transmission, which makes messages (e.g., IoT device events and commands) fail to be delivered to the destinations. Countermeasure Detection and Consistency Algorithm (CDCA) is proposed in [32] as a countermeasure against reactive jamming attacks on IoT networks, which measures the amount of time spent waiting for the channel to be idle and also checks the consistency of packet signal strength and device location. However, the delay attacks neither cause the channel busy nor affect the signal strength of transmitted packets. Thus, CDCA also fails to detect the delay attacks.