IoT Phantom-Delay Attacks: Demystifying and Exploiting IoT Timeout Behaviors

Chenglong Fu[1], Qiang Zeng[2], Haotian Chi[1], Xiaojiang Du[3], Siva Likitha Valluru[2]
[1] Department of Computer and Information Sciences, Temple University, Philadelphia, PA 19122, USA
[2] Department of Computer Science and Engineering, University of South Carolina, Columbia, SC 29201, USA
[3] Department of Electrical and Computer Engineering, Stevens Institute of Technology, Hoboken, NJ 07030, USA
Email: chenglong.fu@temple.edu, zeng1@cse.sc.edu, htchi@temple.edu, xdu16@stevens.edu, svalluru@email.sc.edu

***Abstract*—This paper unveils a set of new attacks against Internet of Things (IoT) automation systems. We first propose two novel IoT attack primitives: *Event Message Delay* and *Command Message Delay* (event messages are generated by IoT devices to report device states, and command messages are used to control IoT devices). Our insight is that timeout detection in the TCP layer is decoupled from data protection in the Transport Layer Security (TLS) layer. As a result, even when a session is protected by TLS, its IoT event and/or command messages can still be significantly delayed without triggering alerts. It is worth highlighting that, by compromising/controlling *one* WiFi device in a smart environment, the attacker can delay the IoT messages of other *non-compromised* IoT devices; we thus call the attacks *IoT Phantom-Delay Attacks*. Our study shows the attack primitives can be used to build rich attacks and some of them can induce persistent effects. The presented attacks are very different from jamming. 1) Unlike jamming, our attacks do not discard any packets and thus do not trigger re-transmission. 2) Our attacks do not cause disconnection or timeout alerts. 3) Unlike reactive jamming, which usually relies on special hardware, our attacks can be launched from an ordinary WiFi device. Our evaluation involves 50 popular IoT devices and demonstrates that they are all vulnerable to the phantom-delay attacks. Finally, we discuss the countermeasures. We have contacted multiple IoT platforms regarding the vulnerable IoT timeout behaviors, and Google, Ring and SimpliSafe have acknowledged the problem.**

I. Introduction

The global IoT market size was valued at $212 billion in 2018 and is expected to reach $1,319 billion by 2026 [59]. Many IoT platforms support the integration of heterogeneous IoT devices and the installation of powerful automation in a smart environment. While prior works have described attacks that exploit implementation vulnerabilities of IoT devices or platforms [27], [30], [37], [53], this work unveils a set of IoT attacks that do not rely on any implementation vulnerabilities but exploit a design flaw in IoT timeout behaviors.

The attacks are built on two *attack primitives*, dubbed *IoT Event Message Delay* (e-Delay, for short) and *IoT Command Message Delay* (c-Delay, for short). An IoT *event* is raised by an IoT device to report the device state usually to an IoT server (such as a "*motion active*" event), and an IoT *command* is typically issued by an IoT server to control a device (such as a "*lock front door*" command). Due to packet transmission time, a delay is inevitable from the moment an event or command is raised to the moment it is delivered. Such delays are typically sub-seconds and usually do not cause issues.

However, our study discovers that attackers can maliciously increase the delay from sub-seconds to minutes or even hours without causing any alarms. As a result, when an IoT cloud server receives an IoT event that actually was raised quite a while ago, it incorrectly assumes the corresponding IoT device state update is fresh. In smart environments, however, outdated knowledge, held by the cyber-world, about the physical world can cause intriguing problems. Thus, this work investigates two critical questions. (1) Why are such large delays possible? (2) What attacks can be built?

To study the first question, we analyze the IoT network protocol stack to demystify IoT timeout behaviors. Our analysis starts from the TCP layer and moves upwards. A key observation is that the timeout detection implemented in the TCP layer is decoupled from the data protection provided by the TLS layer. As a result, an attacker can fool both the IoT device and server to believe that a session is healthy, while the attacker actually delays IoT messages (although the data integrity remains protected by TLS).

Consequently, the allowed delay of event/command messages is only bounded to the timeout behaviors implemented in the application layer, such as MQTT or HTTP. Our investigation shows that despite the diversity of IoT devices, most of them allow c-Delay from multiple seconds to sub-minutes, while the other attack primitive, e-Delay, ranges from sub-minutes to hours.

To explore the second question, we exploit the two attack primitives, e-Delay and c-Delay, to build a family of attacks against smart homes. (They leverage recent advances in inferring smart home privacy by sniffing encrypted packets [17], [21], [42], [57].) While some attacks merely incur automation delays, others can **disrupt, disable**, and **override** automation rules' execution. They are categorized into three types.

Type-I: State-Update Delay Attack. Given a critical IoT device, a home owner should be notified of its state update as soon as possible (e.g., a pop-up notification on her smartphone). However, an attacker can apply e-Delay to packets of an event message that reports an IoT state update. Despite the simplicity of the attack, it can cause severe consequences if the state update from certain IoT devices is delayed for

minutes. Such critical IoT devices include a water sensor that detects room flooding, a smoke detector [49], [55] that reports fires, a contact sensor that detects home invasion, and so forth. In these situations, every second matters.

Type-II: Action Delay Attack. Because of automation, an event can trigger a critical action. Hence, by applying E-DELAY to that event, the triggered critical action is also delayed. For example, a high carbon-monoxide level triggers an open-window action, an attacker can hold the message that reports "*CO high*" to delay the open-window action, which may be fatal to the residents. The attacker can also apply the C-DELAY primitive to the critical action to further delay the action for a longer time.

Type-III: Erroneous Execution Attack. An attacker may leverage the attack primitives to launch two kinds of Erroneous Execution attacks. (1) ***Spurious Execution*** means that an action command that should not be issued is, in fact, issued. For instance, an attacker delays the state-update event that could have turned the automation condition to *false* (hence, the condition remains *true* due to the delay attack); as a result, execution of the rule will trigger a spurious action. (2) ***Disabled Execution*** means an action command that should be issued is not issued. An attacker can apply E-DELAY to the message, which could have turned an automation condition to *true*, until after the trigger event happens, and hence the automation is disabled.

The presented attacks are different from **jamming** (and other Denial of Service attacks) in three aspects. 1) Unlike jamming, our attacks do not discard any packets and thus do not trigger retransmission. Repetitive retransmission of packets is suspicious. 2) Our attacks do not cause disconnection or timeout alerts. 3) Unlike reactive jamming, which selectively jams certain packets but relies on specific hardware [23], [47], our attacks can be launched from an ordinary WiFi device. Thus, while reactive jamming, as a low-level attack method, can be used to implement the delay attacks revealed in this paper, we present an attack method that can be launched using an ordinary device. As an attacker can control one device to attack other non-compromised IoT devices, we call it the *phantom-delay* attack .

We measure the timeout behaviors and delay ranges of 50 popular IoT devices. The allowed delays range from subminutes to hours, which is a considerable time window for attackers. The results alarm that numerous IoT devices can be exploited. (They have the most significant impact on Apple's smart home users, as the HomeKit Accessory Protocol [15] allows event messages to be delayed with an infinite upper bound.) Moreover, through proof of concept exploits, we demonstrate how an attacker can leverage the primitives to launch sophisticated attacks in a real-world environment.

This work studies a largely omitted topic—*IoT timeout behaviors*. It demonstrates that current IoT timeout behaviors are exploitable in cyber-physical systems like smart homes, and reveals a critical design flaw in many WiFi-based IoT devices: *the network-delay detection in the TCP layer is decoupled from the data protection in the TLS layer*. Our study thus raises a *bigger* question: Is TCP+TLS, despite being a cornerstone of Internet security nowadays, really suitable for IoT devices?

We make the following contributions:

- We study IoT timeout behaviors by analyzing the IoT network protocol stack of WiFi-based IoT devices. Results of our analysis show that timeout behaviors among IoT devices are loosely defined and lack standardization.
- Based on the timeout behaviors, we design two attack primitives, E-DELAY and C-DELAY, allowing an attacker to cause significant message delays, without raising alarms in any layers of the IoT network protocol stack.
- Leveraging the two attack primitives, an attacker can cause not only delays of IoT events and commands (Type-I and II attacks) but also spurious or disabled IoT operations (Type-III attack). These attacks do *not* rely on any implementation vulnerabilities but can impose serious threats on smart environments.
- Our evaluation of 50 IoT devices and various IoT platforms shows that the exploitable timeout behaviors *widely* exist. We also demonstrate proof-of-concept attacks that exploit the delays to manipulate smart home automation. The study reveals a critical design flaw of IoT devices built on TCP+TLS.
- We have contacted multiple IoT vendors to report the attacks. Google, Ring, and SimpliSafe have acknowledged the problem and are fixing it.

The rest of the paper is organized as follows. We discuss some IoT background in Section II and present our attack model in Section III. We demystify IoT timeout behaviors in Section IV and build attacks that exploit them in Section V. The evaluation results are presented in Section VI. We discuss potential countermeasures in Section VII and related work in Section VIII. The disclosure plan is described in Section IX. Finally, we conclude in Section X.

II. BACKGROUND

A. IoT Servers

Based on their locations, IoT servers can be roughly categorized into two types: (1) *Cloud IoT servers*, such as those of SmartThings and Amazon, host IoT services on cloud servers. (2) *Local IoT servers*, such as Apple's HomeKit hub, operate on local devices. The cloud IoT servers can be further categorized into *endpoint servers* and *integration servers*. An endpoint server is operated by an IoT device vendor and directly interacts with its own devices. An integration server, on the other hand, indirectly interacts with third-party IoT devices through their endpoint servers using cloud-to-cloud communication. Some cloud-based servers combine the two functionalities. For example, SmartThings's IoT servers can both connect IoT devices directly and integrate third-party devices via vendors' endpoint cloud servers.

B. Automation Rules

An increasing number of IoT platforms support user-customized automation programs to allow devices to work

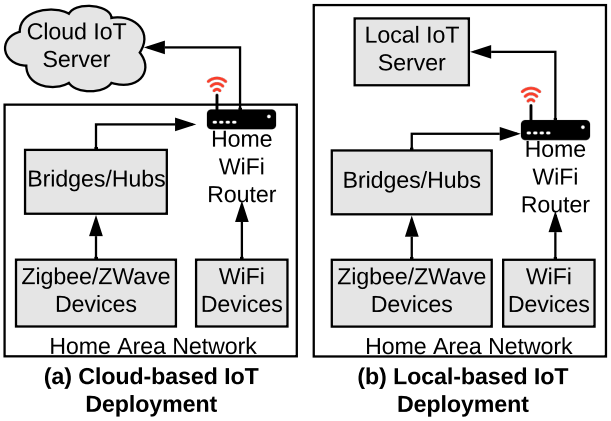

Fig. 1: System model of a typical smart home deployment.

autonomously. Despite the different syntax of automation programs among IoT platforms, an automation program is equivalent to one or more *automation rules*, each of which can be represented as a triplet of ***trigger-condition-action*** (TCA, for short) [24], [28], [58]: When the trigger event is received by an IoT server, if the condition is evaluated as *true*, the specified action is taken. An IoT *event* is usually generated by an IoT device and sent to an IoT server; a delivered event triggers the execution of all automation programs that subscribe to the event. To take an action on an IoT device, an IoT server generates an IoT *command* and sends it to the destination device. Not all rules have *conditions*; in this case, a rule can be regarded as a special case of the TCA rule whose condition is always true [8], [25].

C. IoT Sniffing Based Side Channel Attacks

Recent research has made significant progress in inferring private information of a smart home from its *encrypted* traffic. Some work utilizes the metadata in encrypted traffic, such as packet headers, lengths, and frequencies, to recognize the device's identity [21], [44]. By incorporating the traffic patterns, recent work achieves high precision in inferring events [17], [57] and automation rules [42] from the traffic. Finally, from a sequence of recognized IoT events, patterns of user activities can also be inferred [19], [22], [29], [41], [56]. This work does not advance information inference methods but uses the inferred information to build *active* attacks.

III. System Model and Attack Model

A. System Model

While the presented attacks can probably be applied to other smart environments, to make the discussion and evaluation concrete, this paper mainly considers smart homes. A typical smart home deployment consists of IoT devices, a home WiFi router, and one or more IoT servers. The home router establishes and maintains a home area network and serves for both Internet and LAN access. As illustrated in Figure 1(a), for a *cloud-based IoT deployment*, each WiFi device has an individual network connection with the device vendor's endpoint server. These endpoint servers communicate with an integration server (e.g., SmartThings's server), which stores and executes automation rules. As shown in Figure 1(b), for a *local-based deployment*, IoT devices connect with a local server (e.g., Apple's HomePod) that executes automation programs.

B. Attack Model

The goal of the attacker is to delay IoT messages of a target IoT device, without triggering any timeout alerts or disconnection. This study focuses on IoT devices/hubs/bridges that use the TCP/IP suite. Note that **Zigbee/ZWave devices also need an IoT hub/bridge that users TCP/IP to communicate with IoT servers.** For example, Philips Hue lights are connected to a Hue bridge via Zigbee communication, and the bridge maintains TCP connections with the Hue cloud server. Thus, by delaying the TCP sessions of the bridge, an attacker can delay IoT messages of Zigbee/ZWave devices.

To launch phantom-delay attacks, we assume the attacker has the following capabilities. The attacker is able to control one WiFi device in the victim's smart environment, and use it to sniff WiFi traffic and hijack the TCP session of the target device. We then interpret the assumptions.

First, in many environments, such as hotels, offices, and factories, it is very common that users share a WiFi network. Given an attacker (such as a hotel customer) having access to such a network, he can trivially control his own device to launch attacks. Alternatively, given the many vulnerable IoT devices, an attacker can compromise one and use it as a *stepping-stone* for sniffing and delaying the IoT messages of other devices. Rampant IoT attacks, such as Mirai [38], have compromised millions of IoT devices. What makes our attacks unique is that, by compromising one IoT device, the IoT messages of non-compromised devices can be delayed without causing timeout or disconnection alerts.

Second, once the attacker has controlled one WiFi device, he can use it for sniffing. As described in Section II, IoT sniffing is highly accurate in identifying device and message types based on encrypted traffic [17], [20], [21], [57].

Third, the attacker uses the controlled device, called *TCP hijacker*, to hijack the TCP session of the target IoT device. For instance, ARP spoofing [60] is a well-known and mature attack method for TCP session hijacking. A large-scale study [36] shows that IoT devices are widely vulnerable to ARP spoofing, which is confirmed by our evaluation results.[1]

In sum, given an attacker who has controlled an IoT device in the victim smart environment for sniffing and TCP session hijacking, he can launch phantom-delay attacks against other non-compromised IoT devices. For example, a hotel customer may attack the IoT devices of other customers; a student can attack the IoT devices of faculty members and other students who share a WiFi network; a remote attacker who has compromised one device of a home can delay IoT messages to manipulate other devices.

[1]Alternative to ARP spoofing, if the network cable is exposed to an attacker, he can deploy an active TAP [34] to hijack TCP sessions. Moreover, if an attacker has compromised/controlled an ISP router, he can launch the presented attacks at scale.

Clarification I. One misconception is that an attacker who has hijacked a TLS-protected TCP session can perform powerful attacks. Note we assume the attacker does not know the TLS session keys and the TLS implementation has no vulnerabilities. Thus, any attempts trying to forge, modify, discard, or disorder packets protected by TLS will be detected and cause alerts, while phantom-delay attacks delay IoT messages and disrupt automation without causing any alerts.
Clarification II. It is true that if an attacker does not know the traffic fingerprint [57] of an IoT device, the step of sniffing cannot recognize it. But it is worth highlighting that it is unnecessary for an attacker to recognize all the IoT devices. A reasonable attacker can first profile popular IoT devices and uses the knowledge to online recognize them in a victim environment. For example, in the market of home security cameras, the top five brands take more than 30% of the market share [45]. Considering the total sales of 42.5 million home security cameras in 2020, this means attackers can recognize more than 12.5 million home security cameras by only profiling a few cameras. In some cases when an attacker can visually see the IoT device, e.g., in a hotel, he can specifically collect the fingerprint for the device before launching attacks.

Similarly, an attacker does *not* need to hijack the sessions of all IoT devices; instead, a reasonable attacker will hijack the session of an IoT device only if he is interested in delaying its messages.

IV. Demystifying IoT Timeout Behaviors

In this section, we analyze IoT network protocols in different layers regarding their timeout behaviors. Our insight is that, the timeout detection in the TCP layer is decoupled from the data protection in the Transport Layer Security (TLS) layer, which allows attackers to delay messages without breaking data integrity in the TLS layer. Based on this insight, we present a practical method to delay IoT messages without triggering timeout at any layers. The attack method leads to our two attacking primitives: Command Message Delay (c-Delay) and Event Message Delay (e-Delay).

A. IoT Network Protocol Analysis

In Figure 2, we present a typical IoT network protocol stack. The layers beneath the transport layer are not end-to-end and do not impose barriers on attempts to delay IoT messages. Thus, our analysis starts from the transport layer and moves upwards.

1) Transport Layer Protocols: For the transport layer, out of the popular UDP and TCP protocols, we focus on the latter one because the former is rarely adopted for communicating with IoT servers [18], [39] and does not provide any timeout detection. As a connection-oriented protocol, TCP is designed to handle delayed, out-of-order, and lost segments by requiring each transmitted segment to be acknowledged by the receiver. For this purpose, either side of a TCP connection maintains a *retransmission timer* and a *keep-alive timer* [52]. By default, if the ACK is not received before the retransmission timer expires, the sender will make several attempts to re-transmit the packet (with random backoff intervals). If all retransmission attempts fail, the sender will terminate the TCP connection and notify upper-layer protocols of the timeout. Besides, a TCP probing segment is sent if a session stays idle for a period that makes the keep-alive timer expire. Since the TCP ACK is not encrypted and is independent of the segment payload, attackers that have access to the TCP connection can avoid TCP timeout by generating fake ACKs for either normal or probing segments, which fools both sides to believe the connection is still healthy.

2) Transport Layer Security Protocol: The TLS protocol is adopted by most of today's WiFi-based IoT devices to provide security services [18]. An implicit sequence number is maintained by two parties of a TLS session and incremented after each successful transmission. The sequence number is included in the generation of message authentication code (MAC) for each record, which is protected by the TLS session key. Without the TLS session key, attackers cannot forge a valid MAC for disordered or replayed records. However, TLS does not provide timeout detection.

3) Application Layer Protocols: In the application layer, device vendors have the flexibility to choose existing protocols or build their proprietary protocols. Message Queuing and Telemetry Transport (MQTT) and Hypertext Transfer Protocol (HTTP) are two of the most commonly used protocols by IoT devices [18], [39].

The MQTT protocol, which is one of the most popular protocols for IoT devices [37], requires a long-live session between a device and an IoT server, for full-duplex message pushing and subscribing [9]. The protocol requires the IoT device to send *PINGREQ* messages if the connection stays idle for a time longer than a predefined *keep-alive interval*. If a PINGREQ message is not received within 1.5 times of the keep-alive period [54], the server should reset the TCP connection with the device and raise a *"device offline"* alarm.

Unlike MQTT, HTTP-based protocols are connectionless and highly customizable [39]. Usually, the sender of an HTTP request waits for the response from the receiver. If the response is not received before the pre-defined period, the sender raises 408 'request timeout' error [31] and drops the session. The pre-defined period is the message's timeout threshold, which is configurable by both the server and the device sides. Besides normal messages that carry events and commands, some devices also exchange keep-alive messages with their server to maintain *long-live* TCP and TLS sessions. Excessive delay or missing keep-alive messages trigger timeout and cause the session to be dropped. In contrast, other devices only establish *on-demand* sessions with their servers upon sending normal messages and terminate the sessions once the transmission is completed.

B. Description of Device Timeout Behaviors

According to our analysis of IoT network protocols, timeout in the TCP layer can be completely avoided with spoofed

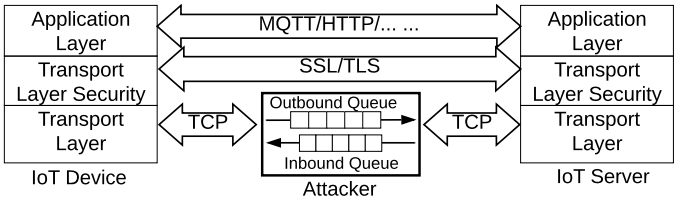

Fig. 2: Delaying IoT messages via a TCP hijacker.

ACKs. Hence, we focus on the timeout behavior at the application layer. We first classify IoT messages into two major types: (i) normal *IoT event/command* messages, and (ii) *keep-alive* (also known as heartbeat) messages that are used to check the connection quality and IoT device or server's liveness. We can describe an IoT device's timeout behaviors using three parameters:

- **Timeout threshold of keep-alive messages**. This parameter is only applicable to devices that use long-live sessions because the purpose of keep-alive messages is to detect and terminate non-responsive sessions.
- **Pattern of keep-alive messages**. This parameter describes what condition will keep-alive messages be exchanged. Keep-alive messages are exchanged either in a *fixed* period or non-periodically when the session stays *idle* longer than a threshold. The keep-alive message pattern comprises the period and strategy (fixed or on-idle) of keep-alive messages.
- **Timeout threshold of normal IoT messages**. This parameter is only applicable to a portion of devices, as some devices do not implement a timeout for event and command messages. For example, MQTT protocol does not require timeout for normal messages.

With the three parameters, an attacker can accurately predict the happening of the incoming timeout of an IoT device, and he can fine-tune the delay imposed on the IoT messages without causing timeouts.

C. Attack Primitives

With the analysis of IoT network protocols, we obtain two major insights: (1) We find that the timeout detection provided by insecure layers (such as TCP) can be fooled. (2) By observing the target device's history traffic, attackers can derive the parameters to model the session timeout behaviors. Based on the parameters, attackers can accurately predict when timeout is to occur and achieve the maximum delay without causing timeout. Following these insights, we build two attack primitives, E-Delay and C-Delay, for delaying event and command messages, respectively.

TCP timeouts can be avoided with forged TCP ACKs. As shown in Figure 2, for each device-to-server TCP connection, via a TCP hijacker, attackers can split the connection between the IoT device and the corresponding cloud server into two separate TCP connections, with the TCP hijacker in the middle. For each side's connection, the attacker can hold the received packets, to incur a delay, before forwarding them to the destination but acknowledge the receiving immediately. If the delay is long and triggers the TCP's keep-alive timeout, the probing segments can be acknowledged using a forged ACK to avoid TCP connection termination. At the end of the delay, all held packets are released in their original order so that the TLS MAC verification is not violated. This way, transport layer protocols no longer have any restrictions on the delay time.

Because of the encryption provided by TLS, we cannot bypass the timeout checking of application layer protocols by forging devices' or server's responses. Instead, we hold target event messages until the moment right before an application layer protocol timeout occurs. During the period of the delay, any following messages are also delayed accordingly to avoid breaking the sequence number checking of TLS.

We present the concrete steps to profile the parameters of a device's timeout behavior. Attackers can perform these steps on their own devices to collect parameters, and then apply them to delay other devices of the same model.

1) By monitoring the device's traffic in an idle state, devices can be distinguished using on-demand sessions by their intermittent TCP/TLS sessions. For devices that are using long-live sessions, the packet length and period of keep-alive messages can be observed.
2) By triggering normal messages of keep-alive devices, a keep-alive pattern can be detected and confirmed. If the next keep-alive message is postponed accordingly, the device's keep-alive messages are exchanged only when the session is idle. Otherwise, the keep-alive messages are exchanged for a fixed period.
3) Timeout threshold of keep-alive messages can be measured by delaying a keep-alive message in an idle state until the timeout happens. The interval between the beginning of the delay and the occurrence of timeout is recorded as the timeout period of keep-alive messages.
4) The timeout threshold of normal (i.e., event and command) messages is measured using the same method as that for keep-alive messages. The message is intentionally triggered right after a successful exchange of a keep-alive message and delayed until a timeout happens. If the timeout occurs earlier than the anticipated timeout of the next keep-alive message (which is also delayed accordingly), this interval is recorded as the timeout of the corresponding message. Otherwise, it means the device does not implement a timeout for normal messages and the session timeout is solely triggered by keep-alive messages.

Finally, the collected parameters can be verified by randomly delaying a message and predicting/observing the timeout behaviors. The parameters are considered to be correct if the predicted timeout matches the real-world timeout.

In summary, the **procedure** for building the attack primitives has the following steps. (1) Before launching the attacks on the victim environment, an attacker selects popular IoT devices as attack targets and profiles their timeout behaviors. Note that the profiling is a one-time effort and the collected knowledge can be shared among attackers. (2) The attacker

sniffs the network traffic in the victim's network and uses the collected knowledge in the previous step to recognize the victim devices. (3) The attacker hijacks the victim IoT devices' traffic to delay IoT messages.

In Section VI-C, we conduct evaluation experiments to measure delay times of 50 popular IoT devices. According to our evaluation results (see Table I and Table II), both c-Delay and e-Delay are feasible on measured devices and allow a delay of dozens of seconds. For most of the devices, e-Delay can be longer than 30 seconds. A straightforward way to understand this is, for example, when a "*smoke-detected*" event arises, our e-Delay attack primitive can delay it for more than 30 seconds without raising any alarms on the device's or server's side.

V. Exploiting IoT Timeout Behaviors

Although the adoption of TLS defeats event and command spoofing attacks, IoT automation systems can still be manipulated using our delay primitives. We propose three types of attacks, based on c-Delay and e-Delay, that can delay critical state updates and actions, override or disable smart home automation, and even incur spurious actions.

A. State-Update Delay Attack

The attack utilizes the e-Delay primitive to delay critical notifications, which defers users' awareness of hazardous situations. For instance, smart home safety monitoring systems have been used in millions of households for providing alerts regarding safety-related incidents [14]. For such IoT devices, *timeliness* is critical as a hazardous situation may occur unexpectedly and develop very fast. Figure 3(a) shows an example where a smart smoke detector is installed in the kitchen. Normally, the "*smoke-detected*" alert is sent to users' smartphones instantly. By applying e-Delay to the smoke detector's event, even for only dozens of seconds, serious damage can be caused when users finally receive the delayed smoke alert.

B. Action Delay Attack

Given an automation rule, by applying e-Delay to its trigger event and/or c-Delay to its action command, the action can be delayed. Some automation rules have critical actions for IoT devices to respond to hazardous situations automatically. For example, as illustrated in Figure 3(b), with a water leak sensor installed in a bathroom, water leaking can trigger the shut-off action of a smart valve. Normally, in the case of a water leak, the valve can be shut off timely, which prevents flooding. By delaying the water leak sensor's event and/or the *shut-off* command towards the valve, the leaking water can cause severe damage.

In particular, if a device is controlled by two separate automation rules that have opposite actions, delaying one action can ***override*** the effect of the other, effectively disordering the two actions. For example, a smart lock that is automatically unlocked when the user's presence sensor becomes on, and then is locked when the contact sensor on the same door turns closed. When the user returns home, by delaying the door-unlock action until after the door-lock action is executed, the door stays unlocked overnight.

C. Erroneous Execution Attack

More than temporarily delaying the device's state-update and actions, the Erroneous Execution Attack imposes more severe safety risks by maliciously invoking or disabling automation rules. Exploiting the delay attack primitives, attackers can selectively delay an IoT event while leaving others untouched. As a result, the delayed event arrives at the IoT server later than other events even if it actually happens earlier in the physical world. The *disordered* events can be used to launch erroneous automation attacks. We first give a formal representation of automation rules that takes delays into consideration, and then describe two types of erroneous automation. Given an automation rule $R = \langle T, C, A \rangle$, where T, C, and C represent the trigger, condition, and action of the rule R. A trigger event instance is denoted as $\mathcal{E}_T$, an event that changes the condition's boolean value $\mathcal{E}_C$, and the evaluation function of the condition C is $f(C)$. The time an event $\mathcal{E}$ is generated by an IoT device is denoted as $I(\mathcal{E})$ and the time it is received by the IoT server is denoted as $S(\mathcal{E})$. When there are no attacks, we assume all events from home to the IoT server use the same transmission time. Thus, if $I(\mathcal{E}_1) < I(\mathcal{E}_2)$, then $S(\mathcal{E}_1) < S(\mathcal{E}_2)$ without attacks. When $\mathcal{E}_T$ is received by the IoT server, if $f(C) = true$, the action A is taken; otherwise, no action is taken. There are two kinds of erroneous execution, discussed as follows.

1) Spurious Execution.: It means an automation rule's condition is falsely satisfied when the rule gets triggered, which leads to an action that should not have been issued in the first place. This can be conducted by applying e-Delay to either the trigger or condition event. (1) Delaying the trigger event $\mathcal{E}_t$. Assume $I(\mathcal{E}_t) < I(\mathcal{E}_c)$; due to the delay attack applied to $\mathcal{E}_t$, $S(\mathcal{E}_c) < S(\mathcal{E}_t)$, and the arrival of $\mathcal{E}_c$ turns $f(C)$ from *false* to *true*. Then, the action of the rule will be spuriously taken when the delayed event $\mathcal{E}_t$ is received. (2) Delaying the event $\mathcal{E}_c$. Assume $I(\mathcal{E}_c) < I(\mathcal{E}_t)$. Due to the delay attack applied to $\mathcal{E}_c$, $S(\mathcal{E}_t) < S(\mathcal{E}_c)$; thus, the action of the rule will be spuriously executed when $\mathcal{E}_t$ is received, as the arrival of $\mathcal{E}_c$ should have turned $f(C)$ from *true* to *false*.

For example, in Figure 3(c), the automation rule is as follows: when a user pulls the storm door, if the user is home, the front door is automatically unlocked. When the user leaves home, attackers can delay the condition event *presence-sensor off* ($\mathcal{E}_c$) while pull the storm door to generate the trigger event *storm-door pulled* ($\mathcal{E}_t$). Because of the delay ($S(\mathcal{E}_t) < S(\mathcal{E}_c)$), $f(C)$ remains $true$ at the time of receiving the trigger event, which causes the front door being unlocked spuriously.

2) Disabled Execution.: Similarly, *disabled execution* can be conducted by delaying trigger or condition events. For example, as shown in Figure 3(d), the automation rules are as follows: when the front door is closed, if the lock is unlocked, the front door is locked automatically. When the user leaves

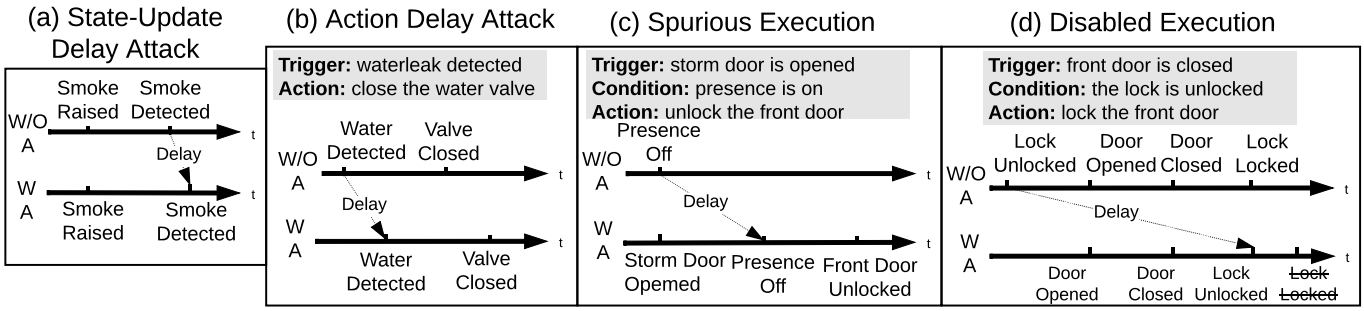

Fig. 3: Example attacks. Two time axes show the sequence of events in the situations, "without attack" and "with attack".

home, he first unlocks and opens the door. However, by delaying the *door-unlocked* event $\mathcal{E}_c$, when the *door-closed* event $\mathcal{E}_t$ is received, the condition evaluation $f(c)$ is still *false*; as a result, the lock-door action is not taken. We denote $\mathcal{E}$ and $\hat{\mathcal{E}}$ as the two opposite events for the same device. This attack can be achieved by either delaying the trigger event $\mathcal{E}_t$ until $f(c)$ is no longer *true*, i.e., $I(\mathcal{E}_t) < I(\hat{\mathcal{E}}_c)$ and $S(\mathcal{E}_t) > S(\hat{\mathcal{E}}_c)$, or delaying the event $\mathcal{E}_c$, which turns $f(c)$ as *true*, to be later than the trigger event (i.e., $I(\mathcal{E}_t) > I(\mathcal{E}_c)$ and $S(\mathcal{E}_t) < S(\mathcal{E}_c)$).

VI. Evaluation

In this section, we evaluate our attack primitives on 50 off-the-shelf commercial IoT devices. For each device, we analyze its timeout behavior and conduct 20-trial experiments to find the range of delay on commands and events. We demonstrate all three types of attacks in real-world testbeds. More details of the attack can be found at our Github Repo [32].

A. Testbed Setup

In our experiment, we test 50 popular IoT devices of 8 types as listed in Table I and Table II. The popularity of each device is indicated by the number of downloads of its mobile app in the Google Play store. All devices are connected to a home Verizon G3100 WiFi 6 router. Low energy devices that do not have WiFi communication capabilities are connected to the router via their compatible hub/bridge devices. A Raspberry Pi 3B is used to simulate an attacker who is in the same local network as the IoT devices.

B. Device Behavior Measurement

First we measure all 50 devices to profile their message characteristics and timeout behaviors. For each device, we use the ARP spoofing to redirect its traffic to the Raspberry Pi, which works as the TCP hijacker shown in Figure 2.

To identify the characteristics of packets that carry the target event or command messages, we first use the server's domain name (for cloud IoT servers) or IP address (for Local IoT servers) to localize the target TCP connection. Then, we trigger events and commands and check the network packets that are received by the raspberry pi. If packets of a certain length can always be found right after the triggering of events or commands, we record it as the length of the target message. Then, we confirm this by repeating the procedure while delaying packets of the same length on the target connection for several seconds. The packet characteristics identification is correct if the anticipated status updates or actions are also delayed by the same amount of time. After that, to verify that the delayed messages are accepted by IoT servers or devices, we use the delayed event to trigger automation rules that have visible actions on other devices (e.g., turning on a light).

For each device, we first set the raspberry pi to the pass-through mode that allows any packet to pass without delay while keeping the device idle to observe its keep-alive period. Then, we conduct a 20-trails experiments for delaying each of the keep-alive, event, and command messages. We have a two-minute interval between every two trials so that the connection of the testing device can resume. We log all TCP segments that flow through our raspberry pi, and analyze them to get the delay behavior parameters by following the procedure as described in Section IV-C.

C. Device Timeout Measurement Results

1) Results of Cloud-based IoT Devices: We first evaluate delays in connections between IoT devices and cloud IoT servers. As shown in Table I, we present their parameters of timeout behavior, which include period and pattern of keep-alive messages, and timeout thresholds for all types of messages. A cell marked as '∞' means we only observe timeouts caused by keep-alive messages among all experiment trials. This indicates that the device does not have timeouts for the event and command messages, and its timeout is solely triggered by keep-alive messages.

From Table I, we can see event messages of all tested devices can be delayed for longer than 30 seconds except the SimpliSafe keypad (HS3), which is enough to cause severe consequences if applied to safety-sensitive events such as smoke alerts. In particular, some WiFi-enabled sensor devices (e.g., M7 and C5) do not use long-live sessions and show delay time windows longer than 2 minutes. Due to the lack of keep-alive messages, a session timeout that is caused by delaying event messages will not be noticed by the cloud server because the session is never established. Even after the session resumes, this anomaly is not reported to the cloud server. Here, we use some example devices to illustrate the procedure to measure the results in Table I.

TABLE I: Measurement results of devices with cloud IoT servers.

No.	Device Type	Device Model	App Install	Long-live Session	Keep-alive Messages			Event Messages		Command Messages	
					Period(s)	Pattern	Timeout(s)	Timeout(s)	Range(s)	Timeout(s)	Range(s)
L1	Smart Light	Wyze White A19	1M+	Yes	62	fixed	60	60	[60, 60]	60	[60, 60]
L2		Philips Hue white A19	1M+	Yes	120	fixed	60	∞	[60, 180]	21	[21, 21]
P1	Smart Plug	Wyze Plug	1M+	Yes	62	fixed	60	60	[60, 60]	60	[60, 60]
P2		Amazon Plug	50M+	Yes	30	fixed	30	30	[30, 30]	30	[30, 30]
P3		SmartThings WiFi Plug	100M+	Yes	110	on-idle	110	∞	[110, 220]	∞	[110, 220]
P4		SmartThings Zigbee Plug	100M+	Yes	31	on-idle	16	∞	[16, 47]	∞	[16, 47]
P5		SmartLife Gosound Plug	5M+	Yes	60	on-idle	32	∞	[32, 92]	∞	[32, 92]
P6		KASA HS103P2 Plug	1M+	Yes	150	fixed	15	55	[15, 55]	15	[15, 15]
P7		Cync	100K+	Yes	21	on-idle	84	∞	[84, 105]	∞	[84, 105]
P8		iHome iSP6X Plug	100K+	Yes	30	fixed	18	32	[18, 32]	32	[18, 32]
P9		Aqara Plug	50K+	Yes	150	fixed	30	60	[30, 60]	30	[30, 30]
P10		Wemo Mini Plug	1M+	No	-	-	-	52	[52, 52]	15	[15, 15]
P11		Geeni Plug	1M+	No	-	-	-	90	[90, 90]	25	[25, 25]
M1	Motion Sensor	SmartThings Motion	100M+	Yes	31	on-idle	16	∞	[16, 47]	-	-
M2		Philips Hue Motion	1M+	Yes	120	fixed	60	∞	[60, 180]	-	-
M3		Wyze Motion	1M+	Yes	62	fixed	60	60	[60, 60]	-	-
M4		Ring Motion	5M+	Yes	30	fixed	35	∞	[35, 65]	-	-
M5		Nest Motion	5M+	Yes	120	on-idle	60	∞	[60, 180]	-	-
M6		Ecobee Smart Sensor	500K+	Yes	60	on-idle	30	∞	[30, 90]	-	-
M7		SmartLife Sonew Motion	5M+	No	-	-	-	260	[260, 260]	-	-
M8		iHome iSB01 Motion	100K+	No	-	-	-	70	[70, 70]	-	-
M9		Aqara Motion	50K+	Yes	150	fixed	30	60	[30, 60]	-	-
M10		Govee Motion	50K+	Yes	90	fixed	35	55	[35, 55]	-	-
M11		Amazon Echo Flex	50M+	Yes	30	on-idle	30	60	[30, 60]	-	-
C1	Contact Sensor	SmartThings multipurpose	100M+	Yes	31	on-idle	16	∞	[16, 47]	-	-
C2		Wyze Contact	1M+	Yes	62	fixed	60	60	[60, 60]	-	-
C3		Nest Contact	5M+	Yes	120	on-idle	60	∞	[60, 180]	-	-
C4		Ecobee Smartsensor	50K+	Yes	60	on-idle	30	∞	[30, 90]	-	-
C5		SmartLife Towode Contact	5M+	No	-	-	-	130	[130, 130]	-	-
C6		iHome iSB04 Contact	100K+	No	-	-	-	70	[70, 70]	-	-
C7		Aqara Contact	50K+	Yes	150	fixed	30	60	[30, 60]	-	-
C8		Ring Contact	5M+	Yes	30	fixed	35	∞	[35, 65]	-	-
C9		Geeni Door & Window	1M+	No	-	-	-	90	[90, 90]	-	-
C10		Govee door	500K+	Yes	90	fixed	35	55	[35, 55]	-	-
HS1	Home Security	Ring Keypad	5M+	Yes	30	fixed	35	∞	[35, 65]	-	-
HS2		Nest Keypad	5M+	Yes	120	on-idle	60	∞	[60, 180]	-	-
HS3		SimpliSafe Keypad	5M+	Yes	55	fixed	30	20	[20, 20]	-	-
S1	Smart Switch	SmartThings button	100M+	Yes	31	on-idle	16	∞	[16, 47]	-	-
S2		Philips Hue Dimmer	1M+	Yes	120	fixed	60	∞	[60, 180]	-	-
S3		ThirdReality Switch	1K+	Yes	31	on-idle	16	∞	[16, 47]	∞	[16, 47]
S4		Aqara Button	50K+	Yes	150	fixed	30	60	[30, 60]	-	-
CM1	Smart Camera	Arlo Q	1M+	No	-	-	-	60	[60, 60]	-	-
CM2		Wyze Cam Indoor	1M+	Yes	62	fixed	60	60	[60, 60]	-	-
CM3		Ring Doorbell	5M+	Yes	55	fixed	25	31	[29, 31]	-	-
CM4		Foscam R2C	1M+	Yes	150	fixed	45	30	[30, 30]	-	-
CM5		YiHome Cam Indoor	1M+	Yes	45	on-idle	30	∞	[30, 74]	-	-
LC1	Smart Lock	August Pro Gen2	500K+	Yes	70	on-idle	30	58	[30, 58]	58	[30, 58]
LC2		Kwikset Smartcode 913	50K+	Yes	31	on-idle	16	∞	[16, 47]	∞	[16, 47]

For devices that are using the SmartThings hub, we start monitoring the hub's network traffic with no device attached. From the long-live TLS session between the SmartThings hub and the SmartThings cloud server, we find that they exchange messages with fixed lengths of 40 bytes (hub to cloud) and 42 bytes (cloud to hub) in every 31 seconds. When an event or command message happens, the next keep-alive messages will always occur 31 seconds later. When we attempt to delay keep-alive messages, we observe a constant timeout threshold of 16 seconds. After that, we trigger and delay event and command messages right after keep-alive messages and find the session timeout still happens 16 seconds after the next keep-alive message. This implies that the SmartThings session timeout is solely triggered by keep-alive messages. We confirm that by randomly triggering and delaying event and command messages and confirmed that timeouts always happen 16 seconds after the starting of delay of a keep-alive message. For Philips Hue Light bulb (L2) and dimmer switch (S2) that are using the Philips Hue bridge, we observe a fixed keep-alive period of 120-second period, which is independent of the event and command messages. During the 20-trial experiment, delays of command messages consistently cause session timeouts after 21 seconds. While, in trials of event message delays, timeout always happens 60 seconds after a keep-alive message, which means there is no dedicated timeout for event messages. In summary, event messages of Philip Hue devices can be delayed in the range of $[60s, 180s]$, which depends on the interval between the event message and the next keep-alive message, and their command messages can be consistently delayed for 21 seconds.

To verify the collected parameters in Table I, we also conduct a verification test. For each testing device, we randomly trigger and delay its messages and predict the timeout occurrence according to the collected parameters. We end the delay and release the holding messages 2 seconds before the predicted timeout. The results show that not only the timeout is 100% avoided, but the delayed messages are also accepted by the device or cloud server.

2) Results of Local-based IoT Devices: For measuring devices' delays with local IoT servers, we choose Apple's HomeKit as the representative IoT server and connect compatible devices, as listed in Table II, to a HomePod speaker using

TABLE II: Measurement results of devices with a local IoT server (a HomePod).

Label	Device Model	Event Messages	
		Max (s)	Min (s)
L2	Philips Hue white A19	420	223
L3	LIFX Mini White A19	412	179
P8	iHome iSP6X Plug	341	115
M2	Philips Hue Motion	290	67
M6	Ecobee Smart Sensor	679	337
M9	Aqara Motion	1310	421
C4	Ecobee Smartsensor	854	211
C7	Aqara Contact	1345	683
S2	Philips Hue Dimmer	275	170
S4	Aqara Button	1453	302
S5	Insignia Garage Controller	343	196
CM1	Arlo Q	200	129

the Home+ mobile application [13] from Apple's App Store. The measurement results show that the HomeKit platform has a much more severer problem. Even though devices are using long-live TCP connections to communicate with the HomePod speaker, we could not find a single keep-alive message when the connection is idle. Moreover, for each event message sent by IoT devices, the local server (i.e., the HomePod speaker) does not give any response, which means that IoT devices have no way to know whether the event messages are received by the server or not. Command and status-polling messages that are issued by the local IoT server have a fixed timeout of 10 seconds for the receiving device to respond. All these findings are confirmed by descriptions in the HomeKit Accessory Protocol (HAP) [15]. This means that the event messages of HomeKit devices can be delayed with a theoretical infinite upper bound. Even if the sporadic command or status polling messages happen during event message delay attacks, attackers can easily avoid timeout as long as they end the delay by releasing all holding packets within 10 seconds. In our 20-trial experiment, the maximum event message delay is over 20 minutes. The delayed events can always be accepted by the local IoT server as valid for triggering the device's state update and automation rules execution, which provides attackers with much longer time windows to launch erroneous execution attacks, and much longer delay for type-I and type-II attacks.

3) Interesting Findings: We present several interesting findings that reveal more design flaws of IoT timeout handling:

Finding 1: Lack of Device Offline Reporting. Although we reasonably assume the "device offline" alarms can be raised when a device's connection with the IoT server is terminated because of the timeout, it is not always the case, according to our experiment. Some long-live connection devices such as Nest security system (HS2), and most on-demand devices (M7, C5, and C9) do not raise a "device offline" alarm immediately after the connection timeout. For the Nest security system, the base station will try to reconnect the server; as long as the reconnecting is successful within three attempts, no alert will be raised. Sensors with on-demand connections (e.g., M7 and C9) do not have offline detection. No matter how many times they are delayed to timeout, these devices do not report the offline events.

Finding 2: Duplicated Connections. On delaying event messages from devices, even after a timeout occurs and the connection closes between a device and our Raspberry Pi, the connection on the other side with the IoT cloud server can remain active, which gives the server an illusion that the device is still online and postpones the "device offline" alerts. After that, the disconnected device will try to reconnect to the server. Surprisingly, even after a new connection has been established, cloud servers still maintain the duplicated half-open connection and do not raise any alerts. Moreover, as long as the new connection is established before closing the half-open connection, the server will never raise the "device offline" alarm. This implies that even timeout happens on an IoT device, attackers can still avoid device-offline alarms by maintaining the half-open connection until the device reconnects to the IoT server. This flaw can be exploited to further extend the length of the stealthy delay.

Finding 3: Unidirectional Liveness Checking. Although most of the tested devices adopt long-live sessions with keep-alive messages, keep-alive requests are always initiated by IoT devices, while IoT servers only passively receive and reply to the keep-alive messages. Even when a device's events are being delayed by attackers, from the perspective of an IoT server, it seems as if the device is just idle. This offers opportunities for attackers to make longer delays on events, until the IoT server can proactively send command messages toward the IoT device.

D. Proof-of-Concept Attacks

We conduct end-to-end experiments in real-world home testbeds with automation rules that are collected from IoT user forums (as listed in Table III). We simulate the attacking device using a raspberry pi that is connected to the participant's home network. We obtained our university's IRB approval for the experiment involving human participants.

1) State-Update Delay Attack: State-update delay attacks are applied to devices' event messages. For Case 1 in Table III, we deploy an Amazon Echo Speaker and a Ring security base station with a contact sensor on the front door. We add a routine that issues voice alerts when the front door is opened. We use Nmap [43] to discover the Ring base station and hijack its TCP connection using ARP spoofing. We locate the target TCP connection with a cloud domain at "*.prd.ring.solution" as carrying event messages. By referring to our measurement results, we can accurately identify messages of keep-alive (48 bytes) and contact sensor (986 bytes), and delay them for up to 60 seconds. It is worth noting that the Ring base station's cellular connection, which could defeat WiFi jamming attacks, is never activated during our attack as the base station is not aware of the attack.

2) Action Delay Attack: We take Case 3 in Table III to demonstrate the action delay attack. We install an August Smart Lock along with the Ring security system into Amazon's Alexa platform and install an automation rule to lock the front door upon door closing. Using a similar method as the state-update delay attack demonstration, we identify the August Lock's connection and message packets. With only

one day's event, we can reasonably infer this automation rule by observing the behavior pattern between the lock's locking commands and the events of door closing. We can proactively verify this hypothesis by adding small delays of five seconds on events of front door closing, and check whether the "door locking" actions are also delayed by five seconds. After that, on detecting "door closed" events, we can delay the following locking command for a period between 30 seconds to 58 seconds. By combining the E-Delay on the contact sensor's event, the length of the attacking window can be extended to be at least 60 seconds, which is enough for burglars to unhurriedly break into a victim's house after the victim leaves home. Moreover, for Case 4, we have another interesting finding that the delayed event messages from the Ring base station, even without causing the base station's disconnection, will be discarded by Alexa if the delay time is longer than 30 seconds. No notification or alert are raised about the lost event. This allows attackers to easily disable the execution of safety-critical routines (e.g., turn off an electric heater when users leave) forever.

3) Erroneous Execution Attack: For Erroneous Execution Attacks, we collect seven real-world conditional automation rules from the IoT user forum (Case 5 to Case 11 listed in Table III), and reproduce them using our testing devices. For the subtype attack of Spurious Execution, we reproduce four cases that aim to make safety-critical actions such as disarming a home security system and unlocking a front door. For example, we reproduce the Case 8 on the SmartThings platform with a SmartThings presence sensor, an August Smart Lock, and a SmartLife contact sensor. Whenever the attacker observes the sequential events of 'storm door opened', 'interior door locked', and 'storm door closed', he can delay the following 'presence off' event by 40 seconds, during which attackers can break in by pulling the storm door.

For the disabled execution attack (Case 9 to Case 11), we take Case 10 as an example. We produce it with an August Smart Lock and a SmartThings contact sensor, which are installed on the same door. When the user leaves home, there should be a clear event sequence of "door unlocked", "door opened", "door closed", and "door locked". Whenever we receive the event of "door unlocked" from the August Connect bridge to the August server, we hold it until the following event of "door closed" is passed to the SmartThings cloud server by the SmartThings hub. According to our measurement, the presence sensor event can be delayed for at least 16 seconds, which is long enough to cover the interval between the door unlocking and closing. We confirm that the door has been left unlocked for the entire day while the participant is away.

VII. Potential Countermeasures

A. Requiring Message ACK and Shortening ACK Timeout

As analyzed in Section IV, the delay time range is closely related to the timeout configuration for application layer acknowledgements. The measurement results in Table I also indicate that a shorter message timeout value can effectively reduce the length of the attack window. According to our measurement, existing IoT devices have an overlong timeout for event message acknowledgement; we suggest shortening it significantly. The HomeKit Accessory protocol does not require acknowledgement of IoT event messages, which leaves an almost unbounded window for our attacks, showing a serious design flaw that should be fixed. The current MQTT protocol does not mandate timeout for an acknowledgement (PUBACK message), but we consider it to be a critical requirement for a secure MQTT implementation.

Limitation: Shortening the keep-alive interval and timeout threshold comes with the cost of increasing device's network traffic. For example, the LIFX light bulb has the keep-alive interval set to less than 2 seconds, which causes more than 150 MB traffic per *hour*. This could cause heavy burdens for a home WiFi network with multiple LIFX bulbs and has been complained by LIFX users [11]. Moreover, for battery-based devices, this countermeasure is not practical. Finally, the application layer timeout threshold needs to take into consideration scheduling, automation processing, and slow devices; thus, it may cause other issues by simply shortening the threshold.

B. Timestamp Checking

Currently, commands and events are accepted if they are generated a long time ago, which shows another critical design flaw of existing IoT protocol stack. We suggest it should be enforced in standard protocols (such as MQTT) by adding timestamps as message fields and checking them at the receiver's side. There should be a mechanism that allows devices and servers to determine whether the received messages are outdated or not.

Limitation: This countermeasure is effective in preventing erroneous execution due to delayed trigger events. However, this solution cannot stop state-update/action delay attacks and erroneous execution attacks due to condition events being delayed. For example, for the attacking Case 8 in Table III, the automation server issues the command according to the automation rule to unlock the door at the time it receives the event of '*storm door opened*'. Even if the server is aware that the latter event of '*presence away*' actually happens earlier than the previous one of '*storm door opened*' and takes the remedial action by relocking the smart lock, the burglar could have already entered the victim's house.

VIII. Related Work

a) Disrupting IoT Message Transmission: There have been many attacks that exploit IoT vulnerabilities [27], [28], [30], [53], [62], [63]. The attacks presented by Hariri *et al.* [35] discard IoT event messages. They discover implementation vulnerabilities of certain IoT devices that allow messages to be intercepted without raising any alarms. OConnor *et al.* [48] explore the possibility of blocking an IoT device's events that are transmitted through on-demand connections. They find that, by blocking the device's event message, the

TABLE III: Cases of event delay attacks. Rules used in these cases are collected from smart home user forums as referred in the last column. In each case, the event being delayed is highlighted in italic font.

Case	Type	Trigger	Condition	Action	Consequence	Reference
Case 1	State-Update Delay	*Front door opened*	-	Voice notification	late burglary alerts	[16]
Case 2		*Motion active*	-	Mobile notification	late burglary alerts	[6]
Case 3	Action Delay	*Front door closed*	-	*Lock the door*	door not locked in time	[12]
Case 4		*Home security system armed*	-	*Turn off heater*	heater not turned off	[12]
Case 5	Erroneous Execution	*Front door unlocked*	Entrance motion inactive	Disarm security system	security system disarmed	[7]
Case 6		*Bedroom motion active*	Bedroom door closed	Turn on bedroom heater	heater maliciously turned on	[5]
Case 7		*Study motion active*	Study door closed	Open the study window	window maliciously opened	[2]
Case 8		Storm door opened	*Presence on*	Unlock the interior door	door maliciously unlocked	[3]
Case 9		Presence away	*Front door open*	Send text message	door open notification muted	[4]
Case 10		Front door closed	*Front door unlocked*	Lock the front door	door not locked	[1]
Case 11		Presence away	*Heater is on*	Turn off Heater	heater not turn off	[10]

event gets lost permanently. However, attacks proposed by these works are only applicable to certain devices that have specific implementation vulnerabilities. Attacks in [35] are only applicable to devices with defective TLS implementations that do not take the TLS record sequence number into consideration. As admitted by the authors, on devices with a secure implementation, their attacks will result in connection termination and device-offline alarms immediately. Attacks in [48] only work on devices that use on-demand connections or have their heartbeat and event messages transmitted through different sessions. According to our evaluation, most IoT devices use long-live connections and have their heartbeat and normal event messages transmitted through the same TCP session. Both attacks may be detected as IoT anomalies of missing IoT messages [33]. Unlike these works, our attacks do not depend on any implementation vulnerabilities and are widely applicable to existing WiFi IoT devices.

b) WiFi Jamming Attacks.: Traditional jamming attacks block the radio frequency channel (by sending strong noise signals) or send certain packets to cause disconnections. This usually raises alarms due to disconnections. Reactive jamming attacks generate radio bursts to reactively suppress the transmission of interested packets [40], [46], [47], [61]. Reactive jamming attacks are more stealthy, but are difficult to launch due to the strict requirement that target frame detection and jamming be finished within hundreds of nanoseconds. As a result, the attacks usually require special hardware [50]. While jamming may also be used to implement phantom-delay attacks, we present a more stealthy attack method that can be launched from an ordinary WiFi device.

c) Delay Attacks: Delay attacks have been well-explored in distributed systems and wireless sensor networks. Bianchin *et al.* [26] model and analyze the relationship between the distribution degree of a distributed system and its robustness against delay attacks. Ranganathan *et al.* [51] survey a list of time-delay attacks in wireless sensor networks. These works focus on theoretical modeling of the effect of message delay attacks in a distributed system. However, they do not discuss how to stealthily launch the attack in real systems. In comparison, our phantom-delay attack is based on a thorough study of timeout behaviors of IoT devices, and our work is able to stealthily delay messages in real IoT systems, without triggering any timeout alerts.

IX. Vulnerability Disclosure

We have filed vulnerability disclosure reports to security teams of Apple HomeKit, Samsung SmartThings, Google Nest, Amazon Ring, and SimpliSafe, ensuring that the disclosure is at least 90 days before the publication. In these reports, we explain the risk of long timeout and keep-alive periods and provide a detailed description and code for reproducing the message delay attacks. Google, Ring, and SimpliSafe have acknowledged our reported vulnerability and made plans to mitigate the problem. Our request of CVEs is being processed and the update will be posted at our GitHub repo [32].

X. Conclusion

This work studies an important but largely omitted topic—IoT timeout behaviors. We have proposed novel attack primitives: an attacker who leverages mature attack methods, such as sniffing and ARP spoofing, can delay IoT messages significantly without triggering alarms in any layers of the IoT network protocol stack. With the attack primitives, we constructed attacks that can delay critical state updates or maliciously disable/enable/delay/override automation actions. The evaluation of 50 popular IoT devices shows that the attacks are widely applicable to IoT devices, and we demonstrated a variety of PoC attacks. Our future work will investigate exploitable timeout behaviors in other IoT protocols and propose defenses against the delay attacks.

This study reveals a critical design flaw in many WiFi-based IoT devices: *the network-delay detection in the TCP layer is decoupled from the data protection in the TLS layer.* Moreover, as the application layer timeout threshold needs to take into consideration scheduling, automation processing, and constrained devices, simply shortening the threshold may cause other issues. Given the design flaw and the dilemma in the application-layer timeout design, we thus raise a *bigger* question: Is TCP+TLS, despite being a cornerstone of Internet security nowadays, really suitable for IoT devices?

Acknowledgment

This work was supported in part by the US National Science Foundation (NSF) under grants CNS-1828363, CNS-2204785, CNS-2205868, CNS-1856380, CNS-2016415, CNS-2107093, and CNS-2144669. The authors would like to thank anonymous reviewers and our shepherd, Dr. Haining Wang.

References

[1] "Automation: Front door lock locking when its not supposed to," https://community.smartthings.com/t/automation-front-door-lock-locking-when-its-not-supposed-to, 2013.

[2] "Climate control guru," https://community.smartthings.com/t/climate-control-guru-beta, 2017.

[3] "Door unlock automation? how do you reduce security risks?" https://community.smartthings.com/t/door-unlock-automation-how-do-you-reduce-security-risks, 2018.

[4] "Opening door lock with mobile presence: good idea or not?" https://community.smartthings.com/t/opening-door-lock-with-mobile-presence-good-idea-or-not/, 2018.

[5] "Piston in a stuck state," https://community.webcore.co/t/piston-in-a-stuck-state/, 2018.

[6] "Receiving notifications of motion while home and disarmed," 2018. [Online]. Available: https://community.smartthings.com/t/problem-receiving-notifications-of-motion-while-home-and-disarmed/120905

[7] "Routine to arm monitor based on lock and presence?" https://community.smartthings.com/t/routine-to-arm-monitor-based-on-lock-and-presence/, 2018.

[8] "Create your own if this then that rule on IFTTT," https://ifttt.com/create, 2019.

[9] "Google cloud iot protocols," 2019. [Online]. Available: https://cloud.google.com/iot/docs/concepts/protocols

[10] "Have i done it? - thermostat/motion for space heater," https://community.webcore.co/t/have-i-done-it-thermostat-motion-for-space-heater/, 2019.

[11] "Lifx bulb uploading a lot of data," 2019. [Online]. Available: https://community.lifx.com/t/lifx-bulb-uploading-a-lot-of-data/6081

[12] "Arming or disarming your ring alarm with your smart lock," 2020. [Online]. Available: https://support.ring.com/hc/en-us/articles/360022350352-Arming-or-Disarming-Your-Ring-Alarm-with-your-Smart-Lock-

[13] "Home+ 4," 2020. [Online]. Available: https://apps.apple.com/us/app/home-4/id995994352

[14] "Home security systems market by home type, security, systems, services, region - global forecast 2025," 2020. [Online]. Available: https://www.reportlinker.com/p05495954/Home-Security-System-Market-by-Home-Type-System-Type-Offering-And-Geography-Global-Forecast-to.html?utm_source=GNW

[15] "Homekit," 2020. [Online]. Available: https://developer.apple.com/homekit/specification/

[16] "Ring security system integration with echo devices," 2020. [Online]. Available: https://www.reddit.com/r/Ring/comments/dw6eoo/ring_security_system_integration_with_echo_devices/

[17] A. Acar, H. Fereidooni, T. Abera, A. K. Sikder, M. Miettinen, H. Aksu, M. Conti, A.-R. Sadeghi, and S. Uluagac, "Peek-a-boo: I see your smart home activities, even encrypted!" in *Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, 2020, pp. 207–218.

[18] O. Alrawi, C. Lever, M. Antonakakis, and F. Monrose, "Sok: Security evaluation of home-based iot deployments," in *2019 IEEE symposium on security and privacy (sp)*. IEEE, 2019, pp. 1362–1380.

[19] N. Apthorpe, D. Y. Huang, and D. Reisman, "Keeping the smart home private with smart (er) iot traffic shaping," *Proceedings on Privacy Enhancing Technologies*, vol. 2019, no. 3, pp. 128–148, 2019.

[20] N. Apthorpe, D. Y. Huang, D. Reisman, A. Narayanan, and N. Feamster, "Keeping the smart home private with smart (er) iot traffic shaping," *arXiv preprint arXiv:1812.00955*, 2018.

[21] N. Apthorpe, D. Reisman, and N. Feamster, "A smart home is no castle: Privacy vulnerabilities of encrypted iot traffic," *arXiv preprint arXiv:1705.06805*, 2017.

[22] N. Apthorpe, D. Reisman, S. Sundaresan, A. Narayanan, and N. Feamster, "Spying on the smart home: Privacy attacks and defenses on encrypted iot traffic," *arXiv preprint arXiv:1708.05044*, 2017.

[23] E. Aras, N. Small, G. S. Ramachandran, S. Delbruel, W. Joosen, and D. Hughes, "Selective jamming of lorawan using commodity hardware," in *Proceedings of the 14th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, 2017, pp. 363–372.

[24] J. Bae, H. Bae, S.-H. Kang, and Y. Kim, "Automatic control of workflow processes using ECA rules," *IEEE transactions on knowledge and data engineering*, vol. 16, no. 8, pp. 1010–1023, 2004.

[25] I. Bastys, M. Balliu, and A. Sabelfeld, "If this then what?: Controlling flows in iot apps," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2018, pp. 1102–1119.

[26] G. Bianchin and F. Pasqualetti, "Time-delay attacks in network systems," in *Cyber-Physical Systems Security*. Springer, 2018, pp. 157–174.

[27] H. Chi, Q. Zeng, X. Du, and J. Yu, "Cross-app threats in smart homes: Categorization, detection and handling," *arXiv preprint arXiv:1808.02125*, 2018.

[28] ——, "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, 2020, pp. 411–423.

[29] B. Copos, K. Levitt, M. Bishop, and J. Rowe, "Is anybody home? inferring activity from smart home network traffic," in *2016 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2016, pp. 245–251.

[30] E. Fernandes, J. Jung, and A. Prakash, "Security analysis of emerging smart home applications," in *2016 IEEE symposium on security and privacy (SP)*. IEEE, 2016, pp. 636–654.

[31] R. Fielding and J. Reschke, "Rfc 7231-hypertext transfer protocol (http/1.1): Semantics and content," *Internet Engineering Task Force (IETF)*, 2014.

[32] C. Fu, Q. Zeng, H. Chi, X. Du, and S. L. Valluru, "IoT-Phantom-Delay-Attack," 2022, https://github.com/infinitywings/IoT-Phantom-Delay-Attack.

[33] C. Fu, Q. Zeng, and X. Du, "HAWatcher: Semantics-Aware Anomaly Detection for Appified Smart Homes," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 4223–4240.

[34] Gigamon, "Understanding network taps – the first step to visibility," 2016. [Online]. Available: https://www.gigamon.com/resources/resource-library/white-paper/understanding-network-taps-first-step-to-visibility.html

[35] A. Hariri, N. Giannelos, and B. Arief, "Selective forwarding attack on iot home security kits," in *Computer Security*. Springer, 2019, pp. 360–373.

[36] D. Y. Huang, N. Apthorpe, F. Li, G. Acar, and N. Feamster, "Iot inspector: Crowdsourcing labeled network traffic from smart home devices at scale," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 4, no. 2, pp. 1–21, 2020.

[37] Y. Jia, L. Xing, Y. Mao, D. Zhao, X. Wang, S. Zhao, and Y. Zhang, "Burglars' iot paradise: Understanding and mitigating security risks of general messaging protocols on iot clouds," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 465–481.

[38] G. Kambourakis, C. Kolias, and A. Stavrou, "The mirai botnet and the iot zombie armies," in *MILCOM 2017-2017 IEEE Military Communications Conference (MILCOM)*. IEEE, 2017, pp. 267–272.

[39] D. Kumar, K. Shen, B. Case, D. Garg, G. Alperovich, D. Kuznetsov, R. Gupta, and Z. Durumeric, "All things considered: an analysis of iot devices on home networks," in *28th {USENIX} Security Symposium ({USENIX} Security 19)*, 2019, pp. 1169–1185.

[40] M. Li, I. Koutsopoulos, and R. Poovendran, "Optimal jamming attacks and network defense policies in wireless sensor networks," in *IEEE INFOCOM 2007-26th IEEE International Conference on Computer Communications*. IEEE, 2007, pp. 1307–1315.

[41] X. Liu, Q. Zeng, X. Du, S. L. Valluru, C. Fu, X. Fu, and B. Luo, "Sniffmislead: Non-intrusive privacy protection against wireless packet sniffers in smart homes," in *24th International Symposium on Research in Attacks, Intrusions and Defenses*, 2021, pp. 33–47.

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

[43] G. F. Lyon, *Nmap network scanning: The official Nmap project guide to network discovery and security scanning*. Insecure, 2009.

[44] M. Miettinen, S. Marchal, I. Hafeez, N. Asokan, A.-R. Sadeghi, and S. Tarkoma, "Iot sentinel: Automated device-type identification for security enforcement in iot," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2017, pp. 2177–2184.

[45] J. Narcotta and B. Ablondi, "Strategy analytics: Amazon's ring claimed top spot in global home security camera market in 2020," 2021. [Online]. Available: https://www.businesswire.com/news/home/20210505005395/en/Strategy-Analytics-Amazons-Ring-Claimed-Top-Spot-in-Global-Home-Security-Camera-Market-in-2020

[46] R. Negi and A. Rajeswaran, "Dos analysis of reservation based mac protocols," in *IEEE International Conference on Communications, 2005. ICC 2005. 2005*, vol. 5. IEEE, 2005, pp. 3632–3636.

[47] D. Nguyen, C. Sahin, B. Shishkin, N. Kandasamy, and K. R. Dandekar, "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*, 2014, pp. 15–22.

[48] T. OConnor, W. Enck, and B. Reaves, "Blinded and confused: uncovering systemic flaws in device telemetry for smart-home internet of things," in *Proceedings of the 12th Conference on Security and Privacy in Wireless and Mobile Networks*, 2019, pp. 140–150.

[49] Onelink, "Onelink smoke detector and co alarm," 2021, https://onelink.firstalert.com/.

[50] H. Pirayesh and H. Zeng, "Jamming attacks and anti-jamming strategies in wireless networks: A comprehensive survey," *arXiv preprint arXiv:2101.00292*, 2021.

[51] P. Ranganathan and K. Nygard, "Time synchronization in wireless sensor networks: A survey," *Int. J. UbiComp*, vol. 1, no. 2, pp. 92–102, 2010.

[52] I. RFC793, "Transmission control protocol," *IETF, September*, 1981.

[53] E. Ronen, A. Shamir, A.-O. Weingarten, and C. O'Flynn, "Iot goes nuclear: Creating a zigbee chain reaction," in *IEEE Symposium on Security and Privacy (SP)*, 2017.

[54] O. Standard, "Mqtt version 3.1. 1," *URL http://docs. oasis-open. org/mqtt/mqtt/v3*, vol. 1, 2014.

[55] G. Store, "Nest protect smoke and co alarm," 2021, https://store.google.com/product/nest_protect_2nd_gen?hl=en-US.

[56] A. Subahi and G. Theodorakopoulos, "Detecting iot user behavior and sensitive information in encrypted iot-app traffic," *Sensors*, vol. 19, no. 21, p. 4777, 2019.

[57] R. Trimananda, J. Varmarken, A. Markopoulou, and B. Demsky, "Packet-level signatures for smart home devices," *Signature*, vol. 10, no. 13, p. 54, 2020.

[58] B. Ur, E. McManus, M. Pak Yong Ho, and M. L. Littman, "Practical trigger-action programming in the smart home," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2014, pp. 803–812.

[59] Verified Market Research , "Internet of things (iot) market worth $1319.08 billion, globally, by 2026 at 25.68% cagr: Verified market research," https://www.prnewswire.com/news-releases/internet-of-things-iot-market-worth-1319-08-billion-globally-by-2026-at-25-68-cagr-verified-market-research-301092982.html, 2020.

[60] S. Whalen, "An introduction to arp spoofing," *Node99 [Online Document], April*, 2001.

[61] M. Wilhelm, I. Martinovic, J. B. Schmitt, and V. Lenders, "Short paper: Reactive jamming in wireless networks: How realistic is the threat?" in *Proceedings of the fourth ACM conference on Wireless network security*, 2011, pp. 47–52.

[62] W. Zhang, Y. Meng, Y. Liu, X. Zhang, Y. Zhang, and H. Zhu, "Homonit: Monitoring smart home apps from encrypted traffic," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 1074–1088.

[63] W. Zhou, Y. Jia, Y. Yao, L. Zhu, L. Guan, Y. Mao, P. Liu, and Y. Zhang, "Discovering and understanding the security hazards in the interactions between iot devices, mobile apps, and clouds on smart home platforms," in *28th {USENIX} Security Symposium ({USENIX} Security 19)*, 2019, pp. 1133–1150.