Zigbee's Network Rejoin Procedure for IoT Systems: Vulnerabilities and Implications

Jincheng Wang
jcwang@cse.cuhk.edu.hk
The Chinese University of Hong Kong
Shatin, N.T., Hong Kong

Zhuohua Li
zhli@cse.cuhk.edu.hk
The Chinese University of Hong Kong
Shatin, N.T., Hong Kong

Mingshen Sun
sunmingshen@baidu.com
Baidu Security

John C.S. Lui
cslui@cse.cuhk.edu.hk
The Chinese University of Hong Kong
Shatin, N.T., Hong Kong

ABSTRACT

Internet of Things (IoT) services are gaining increasing popularity, and IoT devices are widely deployed at many smart homes. Among all the IoT communication protocols, Zigbee is a dominant one used by billions of devices and customers. However, the design of Zigbee has not been carefully evaluated and could be exploited by attackers. In this paper, we focus on Zigbee's *network rejoin* procedure, which aims to allow devices to automatically recover their network status when they accidentally go offline. We develop an automated verification tool VERJOIN to perform a systematic study on the rejoin procedure. Using this tool, we not only confirm a well-known design flaw, but also reveal two undiscovered design flaws. Moreover, we construct four proof-of-concept (PoC) attacks to exploit these design flaws. These vulnerabilities create new attack surfaces for attackers to manipulate Zigbee devices, and the damage of these vulnerabilities ranges from denial of service to device hijacking. We further design a Zigbee testing tool ZIGHOMER to confirm these vulnerabilities in real-world devices. Using ZIGHOMER, we conduct thorough evaluations of off-the-shelf Zigbee devices from leading IoT vendors, and the evaluation result shows the prevalence and severity of these vulnerabilities. Finally, we reported our findings to related parties, and they all acknowledged the significant security impact. We further collaborate with Zigbee Alliance to amend the Zigbee specification, and successfully addressed our reported vulnerabilities.

CCS CONCEPTS

• **Security and privacy → Mobile and wireless security**.

KEYWORDS

Zigbee, network rejoin procedure, model checking

RAID 2022, October 26–28, 2022, Limassol, Cyprus

ACM ISBN 978-1-4503-9704-9/22/10…$15.00
https://doi.org/10.1145/3545948.3545953

ACM Reference Format:
Jincheng Wang, Zhuohua Li, Mingshen Sun, and John C.S. Lui. 2022. Zigbee's Network Rejoin Procedure for IoT Systems: Vulnerabilities and Implications. In *25th International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2022), October 26–28, 2022, Limassol, Cyprus.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3545948.3545953

1 INTRODUCTION

Smart homes are prevalent under the rapid growth of the Internet of Things (IoT). IoT devices such as smart hubs, locks, and sensors are widely deployed at smart homes. These devices are usually connected to a wireless network to provide various services. For such wireless networks, the communication protocol is critical because it specifies how devices maintain their network connections and exchange data. Any protocol vulnerabilities can cause unexpected behavior of these devices, and attackers can exploit these vulnerabilities to cause serious damage [38]. Zigbee is one of the most popular IoT communication protocols due to its low power consumption, high scalability, and efficiency [8]. Reports state that more than 3.8 billion Zigbee devices will be sold worldwide by 2023 [9]. Briefly speaking, Zigbee specifies various network procedures for devices to establish and recover their network connections. While these procedures greatly ease device management and control, few investigations evaluate their security guarantees. In this work, we focus on Zigbee's *network rejoin* procedure, which specifies how offline devices recover their network status and re-enable their functionalities automatically. Specifically, we find that Zigbee specifies a *trust center rejoin* procedure, which sacrifices authenticity to achieve easy device management. Such an insecure-by-design procedure usually appeals to the attackers, for they may exploit the vulnerability of the procedure to sabotage numerous IoT devices. However, to the best of our knowledge, there are no formal investigations studying the security guarantee of the rejoin procedure. As a result, a comprehensive security evaluation of Zigbee's rejoin procedure is necessary. In this paper, we aim to initiate the first systematic security study on the network rejoin procedure.

Model checking for Zigbee. Analyzing the Zigbee protocol and the rejoin procedure is a difficult task, and we need to address the following challenges. (1) *Protocol Complexity*: The protocol specifies various device attributes (e.g., the device type) and diverse types of messages. As a result, multiple Zigbee devices can be intertwined together, which exponentially increases the size of the state space.

(2) *Security Properties*: The protocol specification usually states security properties in an abstract and implicit way [30]. As attackers can exploit Zigbee's design flaws to sabotage real-world devices, it is necessary to precisely identify and validate critical security properties which guard normal device behaviors.

In order to address the above challenges, we decide to use the model checking technique [19]. Compared with other techniques which study the procedure's security guarantees (e.g., fuzzing [54]), the model checking involves comprehensive protocol modeling and a formal definition of the security properties. Moreover, it involves a systematic exploration of the possible states of a procedure. As a result, it is well-suited to find intricate errors lurking in the procedure design [42], and it has been widely used to validate security properties of communication protocols (e.g., the 5G protocol [14, 30]). Specifically, we first model the rejoin procedure with respect to an adversarial environment as a finite-state machine (FSM). The adversary follows the Dolev-Yao attacker model [22], which randomly omits, drops, or injects Zigbee messages while respecting certain well-formedness conditions (e.g., the encryption). Then we identify a set of security properties in the technical specification which are critical to devices' normal functionalities, and formalize them. Finally, we design a security checking tool VerEjoin, which can automatically generate the state model of the device's rejoin procedure, and check whether the security properties hold (Section 3). The outputs of VerEjoin are *counterexamples*. Specifically, these counterexamples are state transition sequences of the state machine which cause violations of the security properties.

Security risks. By inspecting the root causes of these counterexamples, we confirm a well-known design flaw and reveal two undiscovered Zigbee's design flaws. (1) The rejoin procedure allows unsolicited devices to deplete Zigbee network resources. (2) Unsolicited devices can exploit the rejoin procedure to modify the property information of legitimate devices. (3) We confirm that the transmission of the network key, which is used for securing Zigbee communications, is weakly protected. To show how attackers exploit these flaws, we further construct four proof-of-concept (PoC) attacks which (1) prevent legitimate Zigbee devices from joining the network, (2) force legitimate Zigbee devices to go offline and disable their functionalities, (3) hijack legitimate Zigbee devices and transfer the control to the attacker, and (4) hijack the whole Zigbee network. The damage is significant, ranging from denial of service (DoS) to device hijacking.

The Zigbee testing tool. In order to verify our reported vulnerabilities and test real-world devices, we further design and implement ZigHomer, a Zigbee testing tool based on the off-the-shelf hardware radio (Section 5). Specifically, ZigHomer provides several fundamental functionalities (e.g., the *reliable packet injection* functionality to guarantee delivery of packets). Based on these functionalities, we design a reconnaissance module, which detects existing devices in the target Zigbee network. ZigHomer further allows users to specify different testing tasks for these detected devices. In particular, we implement our constructed PoC attacks on Zighomer, and use it to conduct evaluations of our reported vulnerabilities on real-world devices. We show that our tool outperforms other existing Zigbee testing tools.

Impacts. Using ZigHomer, we evaluate the impact of our reported vulnerabilities in terms of prevalence, accessibility, and severity (Section 6). Specifically, we find that nearly 72% of certified Zigbee devices are vulnerable to at least one of our reported vulnerabilities. In particular, we use ten real-world Zigbee devices from four leading vendors (e.g., SmartThings and Philips Hue) to confirm these vulnerabilities. We also show that our reported vulnerabilities are accessible: Attackers can use off-the-shelf hardware to easily initiate an attack with little prior knowledge of the target network (e.g., the security information). We further evaluate the severity of these reported vulnerabilities. The evaluation result shows that the number of affected device functionalities (e.g., actuating and sensing) is significant, and the impacts of these vulnerabilities can last for hours, or at times, stay permanent. Finally, we show that user-installed home automation rules can also be affected. Specifically, we use SmartThings automation rules [50] to initiate the evaluation, and the result shows that 63% of the official rules and 71% of the third-party rules will be disabled if attackers exploit these vulnerabilities. We here summarize our contributions.

- **Systematic study on Zigbee's network rejoin procedure.** We are the first to give a systematic security study on Zigbee's network rejoin procedure. Specifically, we develop an automated checking tool VerEjoin to perform the security check. Given the specified security properties, VerEjoin helps us to confirm a well-known design flaw and identify two undiscovered flaws in the rejoin procedure. We also construct four PoC attacks and successfully initiate them on real-world devices.
- **Zigbee testing tool.** We design a Zigbee testing tool ZigHomer, which is used to perform protocol testing on real-world IoT devices. It first initiates the reconnaissance test, which detects existing devices in the target Zigbee network. Then it allows users to easily specify different testing tasks for these detected devices. We show that our tool outperforms existing testing tools.
- **Comprehensive evaluations and impact analysis.** We thoroughly evaluate our reported vulnerabilities for real-world devices. Moreover, we study their influences on home automation rules. Our evaluations show that these vulnerabilities are severe because property security and human life are under huge threat. We further collaborate with the Zigbee Alliance to amend the current and future versions of Zigbee specifications (Section 7).

2 BACKGROUND

2.1 Zigbee Device Properties

Each Zigbee device has two addresses: an extended address and a short address. The former uniquely identifies a device, whereas the latter is assigned when the device joins a Zigbee network. Zigbee specifies three device types: coordinator, router, and end device. An example Zigbee network is shown in Figure 1.

- End devices (e.g., sensors and switches) are resource-constrained devices that cannot form a network or accept devices. However, they can provide diverse functionalities (e.g., sensing and actuating). Most of them are *sleepy end devices*. Specifically, they periodically send the *Data Request* command to their parents (see definitions below). Then they temporarily turn on their receivers to check if there are any

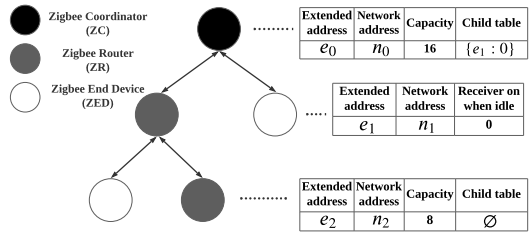


Figure 1: An example Zigbee network

messages from parents. Sleepy and non-sleepy end devices are distinguished by the *receiver on when idle* device property, e.g., in Figure 1, the end device with the address e_1 is a sleepy end device, and its property value is 0.

- The coordinator (e.g., IoT hubs and gateways) is a Zigbee device that can form a Zigbee network, and each network can have only one coordinator. It maintains and manages the network. Specifically, it can accept new end devices, but it has a *capacity* property that specifies the maximum number of end devices it can support. According to the Zigbee specification, the recommended capacity is fourteen. Moreover, it acts as the trust center and it is responsible for distributing the network key. Finally, it maintains a *child table* that records property information of any connected end devices, e.g., in Figure 1, the coordinator records the address and the *receiver on when idle* value for its child device. The child table is also used for managing child devices (e.g., determine the communication pattern according to the *receiver on when idle* property).
- Routers (e.g., light bulbs and smart plugs) can not form a network, but it can accept new routers and end devices. It helps the coordinator to manage end devices. A router also has its capacity limit and maintains a child table.

Because both coordinators and routers can accept end devices, we refer to them as "parent devices" in this paper. Note that Zigbee devices also have diverse application profiles, which specify various device descriptions and functionalities. Popular profiles include *Zigbee Home Automation* (ZHA), *Zigbee Smart Energy* (ZSE) and *Zigbee Light Link* (ZLL). Recently *Zigbee 3.0* is proposed, which unifies these profiles to increase device interoperability. In this paper, we will refer to devices that do not support Zigbee 3.0 as *legacy devices*.

2.2 Network Rejoin Procedure

When an end device accidentally goes offline (e.g., due to a power outage), it will initiate the rejoin procedure to recover its network status. Specifically, the procedure contains the following processes.
Active scanning process. The end device first broadcasts a *Beacon Request* command to locate any parent devices in the network. Then each parent device will reply with a *Beacon Response* command, which tells the end device whether the parent device still has an available capacity. After inspecting all the beacon responses, the end device gets a list of parent devices with available capacity, and will randomly select one as its parent.

Rejoin process. Once the parent is selected, the end device sends a *Rejoin Request* command to the parent, in which its property information is attached, e.g., the *receiver on when idle* property value. Depending on the security level of the request, the rejoin procedure can be either (1) a *secure rejoin* procedure (if the request is encrypted with the network key), or (2) a *trust center rejoin* procedure (if the request is transmitted in plaintext). For the secure rejoin procedure, the parent first checks the encryption of the request. If it is encrypted with the legitimate network key, the parent device accepts the request and sends an encrypted *Rejoin Response* command to the end device. For the trust center rejoin procedure, the parent first accepts the request, then replies with an unencrypted rejoin response. The parent further creates an entry in the child table, and establishes an unauthorized connection with the end device. Note that the end device remains offline unless the connection becomes authorized. As a result, the parent needs to initiate an additional authorization process.
Authorization process. To complete the authorization, the parent first transmits the network key to the end device. The transmission of the network key is encrypted with a symmetric *link key*. After the end device gets the network key, the authorization process is finished, and the end device becomes online. There are several ways to establish the consensus of the link key. Specifically, Zigbee Alliance specifies a *global link key* for all Zigbee devices (i.e., the hex code of "ZigbeeAlliance09"). However, since prior work reveals that the usage of the global link key is vulnerable [36, 48, 52, 56], Zigbee 3.0 proposes a security enhancement that requires each device to use a unique *installation code key* as the link key. Moreover, the link key should be updated once the device joins the network. Finally, vendors can also specify *vendor-specific link keys* for their devices (e.g., Philips Hue specifies a customized ZLL master key [33]). Note that the use of the link key is eventually decided by the trust center [5]. Considering the compatibility of legacy devices, many vendors still support the global link key even for their Zigbee 3.0 devices [4, 6].

3 VEREJOIN

Our goal is to leverage the model checking technique and evaluate the security guarantees of the rejoin procedure. Specifically, the model checking involves a formal model of the procedure and a formal definition of security properties. In this section, we first introduce the detail of procedure modeling and property specification. Then we introduce our security checking tool VEREJOIN, which automatically generates the procedure model and initiates the property checking. Before diving into the details, we first present our threat model.

3.1 Threat Model and Assumptions

In our threat model, the attacker targets legitimate devices deployed by homeowners and legitimate networks maintained by these devices. The attacker has no prior knowledge about the network key. However, one can reasonably assume that the attacker knows the link key, either because the Zigbee network uses the global link key, or because the network uses a vendor-specific link key which has already been exposed (e.g., security analysts extract and release the Hue's ZLL master key by reverse engineer [45]). Additionally, the attacker does not need to physically access the target devices, and she can use off-the-shelf hardware (e.g., the ATUSB [27]) to interact

with the Zigbee communication channel near the victim's home. Specifically, we consider a *Dolev-Yao-style attacker* [22, 30]: The attacker can (1) sniff the Zigbee traffic to extract information, (2) drop or inject any Zigbee message while adhering to cryptographic assumptions, and (3) impersonate a legitimate Zigbee device. In particular, the attacker can send spoofed Zigbee messages and fabricate *phantom devices*, which are non-existing devices or fake devices acting as legitimate devices. For example, the attacker can send a spoofed rejoin request, which fabricates a phantom offline end device.

3.2 Modeling Network Rejoin Procedure

We model the rejoin procedure as a transition system that involves multiple Zigbee devices. Specifically, the rejoin procedure is modeled as a finite-state machine (FSM): $\mathcal{M} = (\mathcal{A}, \mathcal{S}, \mathcal{O}, \mathcal{T}, s_0)$, where $\mathcal{A}$ is a set of actors, and each actor a_i ($i > 0$) represents a specific Zigbee device. Moreover, each actor belongs to a specific device class, i.e., legitimate parent device or legitimate end device. $\mathcal{S}$ is a set of states, and each state $s_t \in \mathcal{S}$ records the data that every actor holds at time t (e.g., the extended address and the network key). In particular, $s_0 \in \mathcal{S}$ is the initial state. $\mathcal{O}$ is a set of operations that actors can take. Each operation can change actors' data and cause the state transition of the state machine. $\mathcal{T} : \mathcal{S} \times \mathcal{O} \rightarrow \mathcal{S}$ is the transition function that drives the system $\mathcal{M}$ to transit from one state to the next state.

To investigate the security guarantee of the rejoin procedure, we further introduce the adversary into the model. Specifically, we expand the supported actor types to include phantom parent and end devices. Since our threat model considers Dolev-Yao-style adversaries, these phantom devices support the operation of sniffing, dropping, and injecting Zigbee messages. These phantom devices randomly decide which operations they will take, such that the adversary can choose any arbitrary strategy to target legitimate devices. For each actor, we identify a set of data which are critical for the rejoin procedure and device functionalities (Appendix A). Specifically, they are device properties (e.g., the *extended address* EA), security information (e.g., the *network key* NK), and the *network status* NS. Given the list of data that actors own, we further give the following definitions.

Def. 1 State: Let $P^t_{a_i}$ denote the data set which actor a_i holds at time t. A state $s_t \in \mathcal{S}$ is the union of all the actors' data, or formally, $s_t = \bigcup_{a_i \in \mathcal{A}} P^t_{a_i}$, among which s_0 is the initiate state. For a specific data $p \in P^t_{a_i}$, Let $P^t_{a_i}[p]$ denote the value of p for actor a_i at time t.

Def. 2 Operation: An operation o_t from actor a_i to actor a_j at time t indicates that a_i transmits a Zigbee command to a_j at time t (We denote $T_x(o_t)$ as the sender and $R_x(o_t)$ as the receiver respectively), or a_i blocks the message sent from a_j at time $t-1$.

The formal definition of operations follows the command descriptions in the Zigbee specification and the Dolev-Yao attacker model. In particular, for the transmission operation, we formulate four message types (Appendix A). For example, the operation *beaconRequest* from a_i to a_j implies that a_i sends a Beacon Request command to a_j, which changes a_i's network status P_{a_i}[NS]. Note that some operations may have several subtypes. For example, the *rejoinRequest* operation has two subtypes, which represent the transmission of encrypted and unencrypted rejoin requests respectively. Given a set of actors and operations, actors may perform their operations in various orders, and we define the *execution path* as follows.

Def. 3 Execution Path: An execution path is an ordered sequence of states $V_s = (s_0, s_1, ..., s_m)$ with an ordered sequence of operations $V_o = (o_0, o_1, ..., o_{m-1})$, such that $\forall i \in \{0, ..., m-1\}, \mathcal{T}(s_i, o_i) = s_{i+1}$. Note that a real-world Zigbee network usually has multiple IoT devices. An execution path describes a possible execution sequence for these devices to initiate the rejoin procedure.

3.3 Defining Security Properties

The set of security goals that we aim to check includes *integrity* (e.g., preventing phantom devices from tampering with legitimate devices' properties), *confidentiality* (e.g., the protection of the network key), and *availability* (e.g., the prevention of the denial-of-service attack). Specifically, we first identify critical security properties from the current technical specification [8] and the vendor's security instruction [1–3]. For instance, according to the specification, *the parent device shall look up and update the entries of its child table when it receives Zigbee packets from a specific extended address.* We therefore identify the child table as a critical device property, and validate its integrity by specifying an integrity property against the phantom device. Note that our specified security properties are categorized into *safety* and *liveness* properties [35, 43]. Informally, safety properties stipulate "something bad should never happen", whereas liveness properties require that "something good will eventually happen". We further adopt *assertions* as well as *LTL formulas* [25] to express these properties. Both are widely supported by popular model checking tools such as Spin [28] and nuXmv [15].

Def. 4 Integrity Property: For each legitimate device, its device property data (Table 4) can not be altered by an operation which involves phantom devices. Specifically, let $Pha(a_i)$ = False denote that a_i is a legitimate device. $\forall t, \forall a_i$ whose $Pha(a_i)$ = False, and $\forall p \in P_{a_i}$ which is the device property data of a_i, we express the integrity property as the following assertion

$$assert(\neg(Pha(T_x(o_t)) \vee Pha(R_x(o_t))) \vee ((Pha(T_x(o_t)) \vee Pha(R_x(o_t))) \wedge (P^t_{a_i}[p] = P^{t+1}_{a_i}[p]))),$$

where $\wedge$, $\vee$, and $\neg$ are logical *and*, *or*, and *negation* operators respectively. At each time point, the assertion first checks actors who will send and receive the Zigbee command in operation o_t. If both are legitimate devices, then the integrity property holds. If any of them is a phantom device, then the assertion further checks that all legitimate device property data are not modified by the operation.

Def. 5 Confidentiality Property: In a Zigbee network, the security information (i.e., the network key) of legitimate devices should be the same and can not be manipulated by phantom devices. Also, the network key of phantom devices can not be the same as the one that legitimate devices own. Let k_l and k_p ($k_l \neq k_p$) be the legitimate and phantom network keys respectively, $\forall a_i$, we have

$$[]((Pha(a_i) \wedge P_{a_i}[\text{NK}] = k_p) \vee ((\neg Pha(a_i) \wedge P_{a_i}[\text{NK}] = k_l))),$$

where [] is the *global* operator used in LTL formulas, and NK represents the *network key*. For an LTL formula $[]\phi$ where ϕ is a logical function, it means that ϕ should always hold at each state. For example, $[](\neg Pha(a_i) \wedge P_{a_i}[\text{NK}] = k_l)$ means that no matter how the system state changes, a_i is always a legitimate device which owns the legitimate network key.

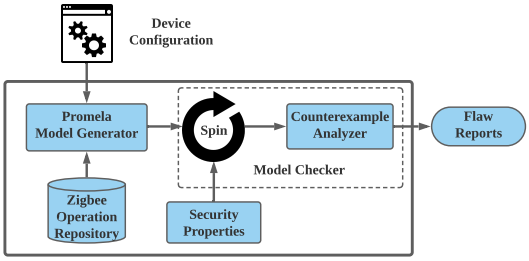


Figure 2: The overview of Verejoin

Note that the confidentiality property is usually defined as a hyper-LTL property [49]. However, considering our settings, it is sufficient to use LTL formulas to specify the confidentiality property. Specifically, hyper-LTL is usually used to validate potential flows of secrets in a system. However, for the rejoin procedure, the flow of the network key is obvious: The key can only flow from the parent device to the end device, and the end device cannot propagate the key anymore. As a result, it is sufficient to only check whether the network key is properly propagated to the end device (i.e., the destination of the transmission), and we achieve this by checking the end device's state using LTL formulas.

Def. 6 Availability Property: Each legitimate end device should eventually be connected to a legitimate parent device and get its network status online. Moreover, since the child table is used for managing child devices, the parent device should maintain correct records in the child table for connected end devices. Specifically, for any legitimate end device a_i, there is a legitimate parent device a_j, such that

$$<> ((P_{a_i}[\text{NS}] = \text{Online}) \wedge (P_{a_i}[\text{PA}] = P_{a_j}[\text{EA}]) \wedge (P_{a_j}[\text{CT}][P_{a_i}[\text{EA}]] = P_{a_i}[\text{RO}])),$$

where <> is the *finally* operator in LTL, and NS, PA, EA, CT, RO represent the *network status*, *parent address*, *extended address*, *child table*, and *receiver-on-when-idle property* respectively. For an LTL formula <> ϕ where ϕ is a logical function, the formula requires that ϕ should eventually hold at some future states. Specifically, the availability property checks whether (1) the network status of a_i is eventually online, (2) the address of a_i's parent belongs to a legitimate parent device, and (3) the parent table contains the record of the child device. In particular, since the child table records the connected end device's address and its RO value, the availability property checks whether the end device's record (index by its address) is properly created and maintained.

3.4 Designing Security Checking Tool

Given the above definition of the model and security properties, we here present the design and implementation of our security checking tool Verejoin, which automates the model generation and property checking. Verejoin has two core components: the *model generator* and the *model checker*, as outlined in Figure 2. The generator takes the user-specified configuration file and pre-defined operations as inputs, and outputs a state machine model for the rejoin procedure. In the configuration file, users configure a list of actors (e.g., their device types), and specify the initial state s_0 for the model. Note that we also provide a default configuration

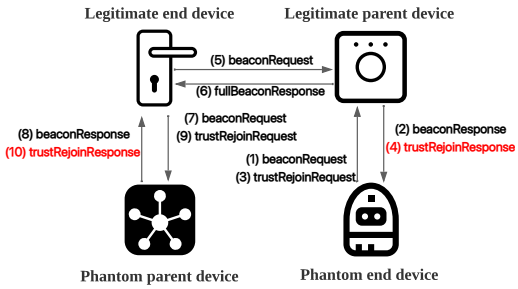


Figure 3: Execution paths violating security properties

file, in which both legitimate and phantom end devices are initially offline. Given the configuration file and the operation template, the generator outputs a state machine model, which describes the states of all actors and their supported operations. Note that the model is specified using the Promela language [39], which can be checked by the model checking tool Spin [28].

We further implement a model checker, which takes the generated model and specified security properties (Section 3.3) as inputs. For each security property, the checker first uses Spin to explore all possible execution paths. When an operation is performed and a new state is generated, Spin checks the security property at the new state. If there is a violation, a trail file is generated which records the detailed state transition. Such a state transition sequence that violates the security property is called a *counterexample*. Our model checker further translates the counterexample into a human-readable operation sequence, and generates a flaw report. Finally, security analysts can check the report to identify the root causes of these counterexamples.

Example. We illustrate the model checking process by an example. Specifically, we use the default configuration file to specify the initial state, and Figure 3 shows reported execution paths that violate the confidentiality property and the availability property. The phantom end device first initiates the trust center rejoin procedure, which is shown in Step (1) to Step (4). Specifically, in Step (2), the *beaconResponse* operation denotes the transmission of the beacon response, which shows that the parent device still has available capacity. In Step (3) and Step (4), the *trustRejoinRequest* and *trustRejoinResponse* denote the transmission of the unencrypted rejoin request and response respectively. Note that the *trustRejoinResponse* operation further checks whether the sender and the receiver own the same link key. If that is the case, the sender will update the receiver's network key. Because all actors initially have the same link key, the phantom device gets the legitimate network key after the operation. Hence, the confidentiality property is violated, and Verejoin reports the above execution sequence. Step (5) and Step (6) show that the legitimate end device also tries to connect to the legitimate parent. However, because the legitimate parent's available capacity has been exhausted, it performs the *fullBeaconResponse* operation, which tells the end device that it is not available. Step (7) to Step (10) show that the legitimate device further initiates the trust center rejoin procedure to connect to the phantom parent device. Finally, the rejoin is successful, and the network key of the legitimate end device is replaced with the

phantom network key in Step (10). Since the above steps violate the confidentiality property and the availability property, VEREJOIN also reports the above execution sequence.

3.5 Checking Result

We first initiate the model checking without the assumption of the well-known link key. Specifically, we enforce the security enhancement of Zigbee 3.0 and each legitimate end device owns a unique link key, which is unknown to the phantom device. As a result, the model checker reports execution sequences which show that (1) phantom devices can consume network resources and prevent end devices from rejoining the network (violation of the availability property), and (2) the end device's record in the child table can be modified by phantom devices (violation of the integrity and availability property). Then we initiate the model checking with the link key assumption. Besides the previously reported sequences, the checker further reports sequences that violate the confidentiality property. Specifically, these sequences show that (3) phantom devices can trigger network key transmissions to leak network keys, and (4) phantom devices can initiate the network key transmission to replace network keys of legitimate end devices.

We further investigate the root cause of these anomalous execution sequences. As a result, we confirm a well-known design flaw and reveal two undiscovered flaws. Finally, we construct PoC attacks based on these sequences, and verify them on real-world devices. In the following section, we will present these design flaws, constructed PoC attacks, and vendors' acknowledgments.

4 REJOIN VULNERABILITIES

In this section, we report three critical design flaws in the Zigbee network rejoin procedure, and these flaws break our specified security properties. Specifically, these flaws are concerned with exploitable network resources, improper protection of device properties, and insecure security material transmissions. We further construct four PoC attacks to exploit these flaws on real-world devices. As a result, the damage of these attacks ranges from denial of service (e.g., triggering online devices to go offline) to device hijacking. All the experiments presented in this work are done ethically: They are conducted only with our own devices in a restricted environment, and we do not put other users or platforms in danger. Finally, we reported our findings to the Zigbee Alliance and corresponding vendors, who all acknowledged the seriousness of these vulnerabilities. Note that we have posted our attack demo videos, captured traffic logs, and alliance responses online for reference [11].

4.1 Exploitable Capacity

Our study discovers that the capacity of parent devices can be exploited, such that both the integrity and availability properties are violated. Specifically, due to the support of the trust center rejoin procedure, the parent device accepts any unencrypted rejoin request unconditionally. For each requesting end device, the parent device establishes an unauthorized connection (Section 2.2) and allocates resources (e.g., the available capacity). As a result, the operation (i.e., the rejoin request) from the phantom device can successfully manipulate the legitimate device's capacity, which implies a violation of the availability property. Specifically, when multiple

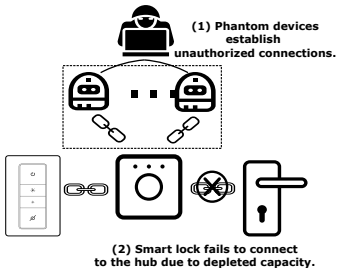


Figure 4: Overview of Capacity Exploitation

phantom end devices exhaust the capacity of a legitimate parent device, other legitimate end devices can not establish connections with the parent device. As a result, the legitimate end device cannot reach the online status and work normally.

Even worse, after checking the specification, we find that the protocol specifies a vulnerable aging-out process for the unauthorized connection. Specifically, Zigbee specifies a timeout for unauthorized connections and requires that the end device reset itself once the timeout is reached. However, the phantom end device will not reset itself. Consequently, once the phantom device establishes the unauthorized connection, the parent device has no choice but to keep maintaining the connection. Therefore, the consumed capacity can not be recovered.

PoC attack: Capacity Exploitation. Given the above vulnerability, we construct a new Zigbee attack: *Capacity Exploitation*. Figure 4 shows the attack overview. Specifically, the attacker creates multiple phantom devices, which send trust center rejoin requests to the legitimate parent device (the hub) and establish unauthorized connections with it. Consequently, these phantom devices deplete the capacity of the hub. Later, when a legitimate smart lock wants to join the Zigbee network, it cannot establish a connection anymore because the capacity of the legitimate parent device has been depleted. The damage of the Capacity Exploitation is threefold. (1) It causes severe damage to offline and freshly new devices, which prevents them from joining the network. (2) It consumes the capacity and computing resources of parent devices for maintaining those unauthorized connections. (3) It helps the attacker to construct more damaging attacks which trigger online devices to go offline (Section 4.2) or hijack legitimate devices (Section 4.3).

We initiate the Capacity Exploitation using eight parent devices from SmartThings, Philips Hue, Xiaomi, and IKEA. The result shows that the attack can fully compromise the capacity of all tested devices. Also, our evaluation shows that the attack can be initiated within seconds, while the influence is persistent. For some victim parent devices, they can never recover their capacities even users initiate the factory reset procedure (see Section 6).

Responsible disclosure. We reported this vulnerability to the affected vendors and Zigbee Alliance, which all acknowledged its seriousness. Specifically, the alliance highlights that we are the first to reveal the insufficient protection of the device capacity and the flawed aging-out process. Moreover, device vendors including SmartThings and Philips Hue indicate that our findings have widespread impacts on many manufacturers' devices.

4.2 Inconsistent Recognition of Device Properties

Our study also reveals that the legitimate devices' records can be easily tampered with, which violates the integrity and availability properties. Recall that the parent device maintains a child table (see Figure 1), in which each entry records property information of a connected end device (e.g., the *address* and the *receiver on when idle* property). Moreover, the parent device determines how to interact with its child device based on the recorded property information. For example, if the recorded *receiver on when idle* value of an end device is 0, the parent device will regard the end device as a sleepy end device. Consequently, the parent will not actively send messages to the end device. According to the Zigbee specification, whenever an end device sends a rejoin request, it should attach its property information in the request. Moreover, once a parent device receives the request, it updates the sender's record according to the attached property information. However, we find that such an update is done without any checking. Specifically, a phantom device can pretend to be the victim end device and send an unencrypted rejoin request with spoofed property information attached. Consequently, the parent device will update the record of the victim end devices using the spoofed property information, which creates inconsistency between the parent and child.

We further discover that the inconsistent recognition of the *receiver on when idle* property can cause victim end devices to go offline. Specifically, for a sleepy end device, once its record is modified, the parent device will mistakenly regard the device as a non-sleepy end device, and change the communication pattern with it: Instead of waiting for a data request from the victim device, the parent device chooses to actively send messages to the victim device, and hopes to get the corresponding acknowledgment (ACK). Since the victim device does not turn on its receiver, it will not receive any message and will not reply with acknowledgment. Consequently, in the parent device's perspective, the victim device seems to lose the message, and the parent will apply a backoff mechanism to retransmit the message. After several failed attempts, the parent device treats the victim device as going offline and automatically disconnects with it. Later, when the victim device sends a data request to poll messages, it will be informed that it has been removed and needs to rejoin the network.

The inconsistent recognition of the *receiver on when idle* property results in the temporary removal of the victim device from the network, and attackers can exploit this vulnerability to temporarily disable the victim device's functionality. However, when the victim device notices that it has been removed from the network, it will immediately initiate the rejoin process to recover its network status. As a result, the offline status is not persistent. We discover that combined with Capacity Exploitation (Section 4.1), the victim device can not recover its network status even if it initiates the rejoin procedure, and we construct the following *Offline Attack*.

PoC attack: Offline Attack. The attack flow is shown in Figure 5, in which the smart lock is a sleepy end device. Specifically, the attacker first initiates the Capacity Exploitation on the target parent device to exhaust its capacity. As a result, the child table has been occupied with the record of phantom devices. After that, the attacker fabricates a phantom end device, which pretends to be the victim lock by sharing the same extended address with it (we will introduce how the attacker gets the address of legitimate devices in Section 5). The phantom device then sends an unencrypted rejoin request to the parent device, in which the *receiver on when idle* property value is set to 1. Consequently, the spoofed request causes an inconsistency issue and disables the regular communication between legitimate devices. As a result, the lock is removed from the network, and its record in the child table is cleared. Note that now the available capacity of the parent device is not depleted anymore due to the removal of the lock. Before the lock notices that it has been removed from the network, the attacker initiates the Capacity Exploitation again to deplete the capacity of the parent device. Later, when the lock tries to rejoin the network, its request will be rejected by the parent device since there is no available capacity. As a result, the lock is stuck in a dangling state: It tries the endless scanning process to find a suitable parent device. Finally, the lock cannot perform its functionalities and cannot be opened by the user.

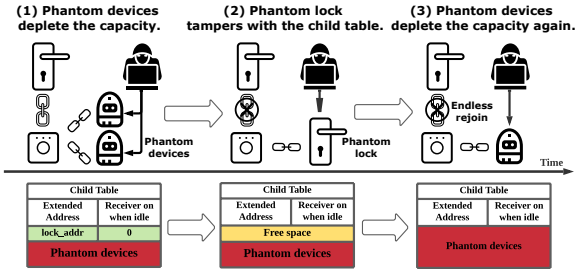


Figure 5: Overview of Offline Attack

We initiate the Offline Attack on eight parent devices from SmartThings, Philips Hue, Xiaomi, and IKEA. Evaluation results show that tested devices from these brands are all vulnerable: Any end devices connected to these parent devices suffer from this attack. Note that this attack disables device functionalities and puts the user's smart home in danger. For example, the offline status of the motion sensor can help the burglar to get rid of the alarm. Consequently, users' belongings can be stolen without detection.

Responsible disclosure. We reported this flaw with constructed attacks to the affected device vendors and the alliance, and they all acknowledged this problem. They confirm that our reported attacks can disable numerous functionalities provided by end devices, e.g., sensing, actuating, and displaying. Since the attack can be combined with the Capacity Exploitation, dangling end devices can even permanently lose functionalities unless users buy a new parent device to accept them. Moreover, the alliance highlights that the manipulation of the device properties by the unauthorized operation is an undiscovered and severe security issue.

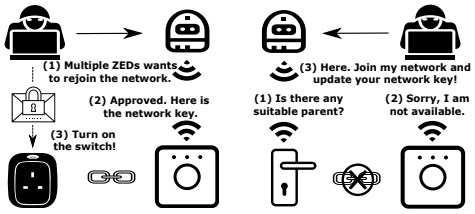
4.3 Support of Well-Known Link Keys

Finally, our study confirms that using the well-known link key for Zigbee devices is vulnerable, and such a design flaw violates the confidentiality property. Recall that the link key is used in the authorization process of the trust center rejoin, and it is used to encrypt/decrypt the transmission of the network key. Since the link key is known by the attacker, the network key transmission

is insecure. In particular, prior work [20, 24, 36, 48, 52, 56] shows that attackers can passively sniff the network key transmission and cause the network key leakage. We highlight that although the vulnerability is well-known, prior work did not construct feasible attacks to exploit it. Specifically, they only focus on the leakage of the network key, and their proposed attacks usually require users' involvement (e.g., wait for users to join devices). We present a detailed discussion at Section 8.

By checking the reported execution sequences that violate the confidentiality property, we construct two PoC attacks that exploit the link key vulnerability. Attackers can initiate these attacks without users' involvement, and we show that the consequence is twofold. (1) A phantom end device can maliciously trigger legitimate parent devices to transmit the encrypted network key. Consequently, the attacker can decrypt it with the well-known link key and cause the network key leakage. (2) A phantom parent device can maliciously initiate the transmission of the network key for offline end devices. Consequently, the attacker can transmit a manipulated network key (encrypted with the link key) and replace the legitimate network key stored in the end device. One can check that the former results in the network key leakage, whereas the latter results in the key manipulation and device hijacking.

PoC attacks: Network Key Leakage Attack and Hijacking Attack. Figure 6a shows the overview of the Network Key Leakage Attack. The attacker first fabricates multiple phantom end devices, each of which has different device properties. In particular, these phantom devices include sleepy and non-sleepy end devices, and their extended addresses are randomly generated. Then these phantom devices send the trust center rejoin request and sniff packets replied by the parent device. They further decrypt these packets with the well-known link key. Finally, if the network key is identified, then the attack succeeds.



(a) Key Leakage Attack (b) Hijacking Attack

Figure 6: Attacks exploiting the link key vulnerability

Figure 6b shows the overview of the Hijacking Attack. Note that a prerequisite of initiating the hijacking is that the legitimate end device should be in the offline status. In order to meet the prerequisite, the attacker can first initiate our constructed Offline Attack (Section 4.2) to induce existing end devices to go offline. As a result, these legitimate end devices go offline and start the scanning process to find suitable parent devices. Then the attacker fabricates a phantom parent device, which broadcasts the beacon response to inform these end devices. The phantom parent device further sniffs packets sent from these offline end devices. Once a trust center rejoin request is identified, the phantom device sends the trust center rejoin response immediately. It further transmits a manipulated network key (encrypted with the well-known link key) to the offline end device. After that, the end device accepts the manipulated network key and sends a broadcast announcement encrypted with the manipulated key. From now on, the end device has been successfully hijacked.

Responsible disclosure. We successfully initiated the attacks on devices from SmartThings, Philips Hue, and Xiaomi. Then we reported the above flaw and PoC attacks to these affected vendors. They acknowledged our findings and deployed the corresponding patches to prevent the attack. We further received a bonus from the Xiaomi bounty program for our successful initiation of the key leakage attack.

4.4 Insufficiencies of Zigbee 3.0

In this section, we introduce Zigbee 3.0's security enhancement of the trust center rejoin procedure [5, 34], and discuss its insufficiencies. Figure 7 shows the enhancement in red fonts. Specifically, the enhancement mainly targets the link key vulnerability. It requires each Zigbee 3.0 end device to update the link key when it joins the network. Later when the end device goes offline and initiates the trust center rejoin procedure, the trust center enforces an additional check for the used link key (by the end device) to determine whether the trust center should transmit the network key to the end device. Unfortunately, the enhancement cannot mitigate our discovered vulnerabilities (Section 4.1 and 4.2). The root cause is that *the enhancement only focuses on the authorization process*. However, our discovered vulnerabilities target the rejoin process, which happens before the authorization process and does not get enhancements. As a result, Capacity Exploitation and Offline Attack are still applicable to Zigbee 3.0 devices.

Even worse, the link key vulnerability (Section 4.3) can still be exploited for Zigbee 3.0 devices. The reason is that the enhancement assumes *all devices in the network should be Zigbee 3.0 devices*, which usually does not hold. Specifically, the enhancement (i.e., the update of the link key) is only possible if both the trust center and the end device support Zigbee 3.0 [5]. Since there are a large number of legacy devices deployed at smart homes, the consequence is twofold. First, when Zigbee 3.0 end devices joins a network formed by legacy parent devices, they still use the global link key or the leaked vendor-specific link key. As a result, attackers can initiate Hijacking Attack to hijack these devices. Second, since legacy end devices do not support the update of the link key, considering the usability of these devices, many Zigbee 3.0 parent devices still support the network key transmission (encrypted with the well-known link key). For example, the SmartThings hub allows users to enable the rejoin procedure with the global link key [6]. As another example, the Silicon Labs' Zigbee stack (i.e., EmberZnet Pro) also keeps the option of using well-known link keys [5]. As a result, these parent devices are still vulnerable to Network Key Leakage Attack.

5 ZIGBEE TESTING TOOL

In this section, we elaborate on the design of our Zigbee testing tool ZigHomer. It is based on ATUSB, an open hardware with the Atmel transceiver [27]. ZigHomer provides fundamental functionalities such as reliable packet injection. Based on these functionalities, it can initiate the reconnaissance test, which detects existing devices

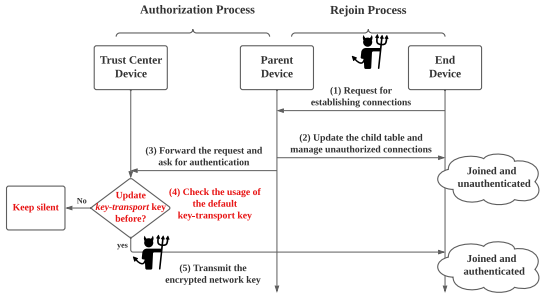


Figure 7: Insufficient security enhancements of Zigbee 3.0

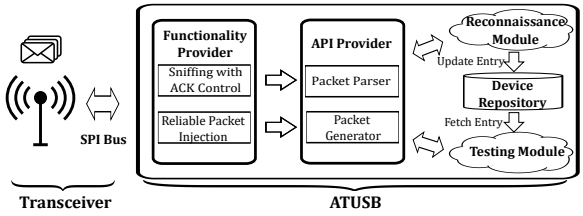


Figure 8: The architecture of ZigHomer

in the target Zigbee network. It further allows users to specify their testing tasks on these detected devices. Specifically, we implement our constructed PoC attacks on ZigHomer to confirm our reported vulnerabilities in real-world devices.

5.1 Architecture Overview

Figure 8 shows the overview of ZigHomer. It is loaded on the ATUSB hardware, and communicates with the transceiver through the serial peripheral interface (SPI). At the bottom, ZigHomer leverages the hardware accelerator of the transceiver to provide two functionalities: *sniffing with ACK control* and *reliable packet injection*. Specifically, the former allows users to sniff Zigbee packets and configure the automated acknowledgment (ACK) reply, whereas the latter allows users to inject Zigbee packets with guaranteed packet delivery. Based on the sniffing and injection functionalities, ZigHomer provides two modules to parse captured Zigbee command packets (the *Packet Parser*) and to inject specific command packets (the *Packet Generator*) respectively. ZigHomer further provides a *Reconnaissance Module*, which detects existing devices in the target Zigbee network and collects their information. The collected device information is stored in the *Device Repository*. Finally, ZigHomer provides a *Testing Module* which loads device information from the repository and performs specified testing tasks on existing devices.

In order to provide the sniffing functionality, ZigHomer first turns on the receiver mode of the transceiver. Later, when a packet arrives, it reads the packet through the SPI bus. Note that according to the Zigbee specification, the receiver should further reply with an acknowledgment to the sender, and there is usually a tight timing window constraint for the transmission of the acknowledgment. For instance, a sender that uses the Q-QPSK physical layer will wait for only 864 microseconds to receive the acknowledgment [13]. To transmit the acknowledgment within the time frame, ZigHomer first configures the transceiver to accelerate the generation of the acknowledgment using its hardware. Then ZigHomer lets users specify a source address such that whenever a Zigbee packet from that address arrives, the transceiver will automatically generate and reply with an acknowledgment.

To provide the reliable injection functionality, ZigHomer first turns on the transmission mode of the transceiver, and passes the packet to the buffer of the transceiver through the SPI bus. Then ZigHomer configures the transceiver to use the CSMA/CA algorithm to avoid the communication channel collision [13]. Specifically, the transceiver first initiates a clear channel assessment (CCA) to check the availability of the current channel. If the communication channel is busy, the transceiver will wait until the channel becomes idle. Later, when the channel is free, the transceiver will transmit the packet, and the algorithm guarantees the successful delivery of packets.

5.2 Packet Parser and Packet Generator

Based on sniffing and injection functionalities, ZigHomer can further parse captured packets and transmit Zigbee command packets. Specifically, the Packet Parser is designed for parsing captured packets. Because ZigHomer may not own the network key used in the target network, the parser can not always capture the packet in plaintext. However, it can still extract useful information from captured packets, i.e., device addresses and command types.

Address information. Zigbee specifies that each network command packet (e.g., the Rejoin Request command) must contain the extended and short addresses in plaintext. For other command packets (e.g., the Data Request command), the format of the addressing filed depends on the specific command type. Therefore, whenever the parser receives a Zigbee command packet, it will parse and extract the address information based on the command type. For each extracted address, the parser further records it in the Device Repository.

Command types. The parser can also identify the command type for a captured packet, and the identification allows ZigHomer to detect and test existing devices (see Section 5.3). Previous research [7] proposed a decision tree to identify network command types for encrypted packets. Our parser implements the decision tree, and can further identify command types for unencrypted packets. Specifically, the parser first extracts the command ID from each unencrypted packet, then identifies the command type according to the Zigbee specification. In total, the parser can identify 23 command types.

The Packet Generator allows users to easily generate Zigbee command packets. Note that for different command types, packets may contain different fields (e.g., the *receiver on when idle* field in the Rejoin Request command). Therefore, the generator provides different APIs to generate different command packets. These APIs also require that users give the address information as inputs. Specifically, users can specify a non-existing address as the source address, which fabricates a non-existing phantom device. Users can also specify an existing address (owned by a legitimate device) as the source address, which fabricates a phantom device pretending to be the legitimate device.

Table 1: Comparisons of Zigbee testing tools

Tool	Available Hardware	Open Source Firmware	Sniffing with Automatic ACK	Reliable Injection	Reconn aissance	Flexible Testing
KillerBee [53]	✓	⋄				✓
Z3sec [41]	✓					
SecBee [56]		✓				
Zigator [7]	✓	✓				
ZigHomer	✓	✓	✓	✓	✓	✓

⋄ means that it depends on the specific hardware.

5.3 Reconnaissance and Testing Modules

It has been shown that the parser can get and record the address information of existing devices. For each recorded address, ZigHomer can further detect its device type by using the Reconnaissance Module. As a result, ZigHomer can learn the overall network topology at target smart homes. Specifically, the Reconnaissance Module first identifies each parent device by exploiting the fact that only parent devices will reply to the Beacon Request command. It first injects a spoofed Beacon Request command packet, then it collects responses sent by existing parent devices and extracts their addresses. After these parent devices are identified, their addresses are labeled as "parent device", and the remaining addresses are labeled as "end device". To further check which parent device an end device is connected to, the module injects a spoofed Orphan Notification command packet. The source address of the spoofed packet is set to be the end device, and this packet is broadcasted to every parent device. Because only the true parent of the end device will reply to this command according to Zigbee specification, the module can learn the parent relationship by checking which parent devices reply to the spoofed command. Finally, the module records learned information about the device type and the network topology in the repository.

ZigHomer also provides an extensible testing module in which users can specify their own testing tasks. Specifically, users can specify a group of devices to be tested (according to the device type) and the interested command type. Then the module fetches related device information from the repository (e.g., the device address), and initiates the injection of the specified command to these devices. The module also allows users to specify a testing routine for these devices instead of a single command type. ZigHomer provides four routines by default, each of which corresponds to a reported PoC attack in Section 4. For example, users can execute the routine of the Capacity Exploitation and specify the parent device as the target. Consequently, for each parent device in the target Zigbee network, the module performs the test to check whether the device's capacity can be exploited. Note that users can easily extend the testing module by specifying their testing routines.

6 EVALUATION

6.1 Comparisons of Zigbee Testing Tools

We choose four existing tools to compare with ZigHomer, and the result is listed in Table 1. Specifically, the *Available Hardware* feature denotes whether the hardware of the tool is still available in the market. At the time of this writing, the hardware of SecBee has been out of date, thus the tool can not be used anymore. The *Open Source Firmware* feature denotes whether the hardware of the tool is open source. Given this feature, users can customize the firmware of the hardware. Tools that do not support this feature (e.g., Z3sec) only provide pre-compiled firmware.

We find that none of the existing tools provide *Sniffing with Automatic ACK* and *Reliable Injection* functionalities. Specifically, these tools fail to support the automated acknowledgment reply, which limits their testing capabilities. Lacking this functionality makes these tools useless when they want to reply to messages that target devices sent. It is also important to note that these tools do not guarantee the successful delivery of packets, which reduces their utilities. We further check whether existing tools support the *Reconnaissance* functionality. Unfortunately, none of them support it. Finally, we focus on the *Flexible Testing* feature, which allows users to extend the tool and specify their testing tasks. We show that only KillerBee and our tool support this feature. One can check that ZigHomer supports all the features mentioned above. The comparison highlights that ZigHomer outperforms existing tools.

6.2 Evaluations of Vulnerabilities

Prevalence analysis. We use ten off-the-shelf devices from four dominant Zigbee device vendors (SmartThings, Philips Hue, IKEA, and Xiaomi) to conduct the evaluation, and Table 2 shows the result. All tested parent devices are vulnerable to Capacity Exploitation and Offline Attack. Moreover, SmartThings Hub and Xiaomi Gateway V2 are vulnerable to the Network Key Leakage Attack. We further find that the Xiaomi gateway V3 uses a vendor-specific link key to prohibit key leakage. However, such a solution has been shown to be insecure: Attackers can get the link key through reverse engineering of the firmware, and the key leakage attack can still be initiated. Finally, we show that the dimmer switch of Philips Hue is vulnerable to the Hijacking Attack. Specifically, when the switch goes offline, it first initiates the secure rejoin procedure. After several failed attempts, it initiates the trust center rejoin procedure and thus can be hijacked. For the multipurpose sensor, because it does not initiate the trust center rejoin procedure when it goes offline (due to the vendor's customization), it is immune to our hijacking attack.

The evaluation result shows that our attacks can fully compromise the capacity and the network status of existing Zigbee devices. The vendors and the alliance also confirmed the prevalence of our reported issues. Specifically, we analyzed 3,188 certified Zigbee devices' information in the official product repository [10], and find that 2,295 of them (71.9%) are vulnerable to at least one of our reported vulnerabilities. For the remaining 893 devices, since they run the Zigbee Smart Energy application profile, which does not support the trust center rejoin procedure by default, we did not count them as affected devices.

Severity analysis. Table 3 shows the time cost and the persistency of the impact for Capacity Exploitation Attack. One can check that it only takes several seconds for attackers to compromise the capacity of affected devices, while it takes hours to recover the capacity. We further find that for parent devices of Philips Hue, the depletion of capacity is permanent. For example, after the capacity of a Hue bridge is depleted, even a firmware update or a factory reset procedure cannot recover its capacity. Note that for the Offline Attack, the above persistency analysis is also applicable. As we previously stated, the offline end device cannot perform any functionalities

Table 2: Evaluation results of real-world devices*

Vendor	Device	Type	Capacity Exploitation	Offline Attack	Key Leakage	Hijacking Attack
SmartThings	Hub V3	ZC	✓	✓	✓	-
	Plug 7APLZJ3	ZR	✓	✓	-	-
	Multipurpose Sensor	ZED	-	-	-	✗
Philips Hue	Bridge V2	ZR	✓	✓	-	-
	Bulb A19	ZR	✓	✓	-	-
	Dimmer Switch	ZED	-	-	-	✓
IKEA	Gateway E1526	ZR	✓	✓	-	-
	Bulb E27	ZR	✓	✓	-	-
Xiaomi	Mijia Gateway V2	ZC	✓	✓	✓	
	Mijia Gateway V3	ZC	✓	✓	✗	-

*The dash symbol means "not applicable". For example, since Capacity Exploitation only targets parent devices, it will not affect ZEDs.

Table 3: Persistency analysis for our reported vulnerabilities

Device	Capacity	Time Cost (s)	Persistency
SmartThings Hub V3	64	3.72	40 minutes
SmartThings Plug 7APLZJ3	15	0.78	40 minutes
Philips Hue Bridge V2	16	0.85	Over 24 hours
Philips Hue Bulb A19	7	0.28	Over 24 hours
IKEA Gateway E1526	32	1.74	2 hours
IKEA Bulb E27	5	0.17	1 hour
Xiaomi Mijia Gateway V2	36	1.83	50 minutes
Xiaomi Mijia Gateway V3	36	1.78	50 minutes

until affected parent devices have the available capacity to accept it. This implies that attackers can disable the end device's functionalities for hours. We further summarize the affected functionalities of certified Zigbee end devices, ranging from actuating (e.g., the remote control and the on/off switch) to sensing (e.g., the motion, smoke, and contact sensors). The invalidation of these functionalities may cause damaging losses to smart home residents. For example, the invalidation of security devices prevents residents from monitoring and guarding their home status. Finally, the vulnerability of the well-known link key allows attackers to hijack end devices and control Zigbee networks. We upload demo videos showing the malicious control online [11].

6.3 Perturbations of Home Automation Rules

Our reported vulnerabilities can also disturb the execution of home automation rules. Nowadays most popular platforms (e.g., SmartThings and IFTTT) allow users to design customized automation rules, and these rules are widely deployed at smart homes or offices. An example is "FireCO_2Alarm" which automatically performs a list of operations (e.g., opening the door) when the CO_2 sensor detects anomalies. In this example, the sensor is the trigger, and the door is the responding device. While the automation rules bring convenience and smart home users heavily rely on them, the evaluation result shows that our reported vulnerabilities can disable these smart rules. Specifically, we choose to use open source automation rules from SmartThings [50] for our evaluation, and we are interested in those rules which specify end devices as the trigger or the responding device. Consequently, a total of 87 official and 36 third-party rules are selected. We further identify four basic classes of impacts and correspondingly summarize the number of affected automation rules. The example rules and evaluation results (Figure 9) are shown in Appendix B for reference.

- *Device functionality interference* denotes that (1) when the rule is triggered, the responding devices cannot function because they have been disabled; (2) the rule cannot be triggered at all because the trigger device has been disabled.
- *Silent notification* denotes that for rules which provide the notification capability, disabling trigger devices prevents these rules from notifying users.
- *Web service interference* denotes that for rules which are designed to provide web services, e.g., weather uploading or cross-platform control services, disabling trigger devices can disable these services.
- *Mode interference* denotes that for rules which can change home modes, disabling trigger devices can disable the transition of the home mode, which affects a list of responding devices.

Our result shows that the *device functionality interference* and the *silent notification* are common impacts on automation rules. Moreover, one automation rule may have multiple functionalities so that they can be affected simultaneously. For example, when the trigger device or the responding device is affected, 14 official rules cannot control the responding device or notify the user. Users' property and lives are in danger if they heavily rely on these automation rules. Even worse, existing security research for automation rules [16, 17, 51] are mostly based on the assumption that IoT devices and the cyber channel are not compromised. Our vulnerabilities break this assumption and bypass existing security systems.

7 DISCUSSION AND FUTURE WORK

Lessons learned. One important lesson learned is that protocol procedures which trade security for utility should be carefully examined. Note that the issue of insecure procedures is not specific to Zigbee protocol. They widely exist in other IoT communication protocols and the functionality implementation of IoT devices. For example, some devices (e.g., routers and cameras) support customized servers to access the device using browsers. However, these servers usually have weak authentication procedures, which allows attackers to easily get the root privilege of these devices [32, 37]. We highlight that model checking technique is powerful for revealing the underlying vulnerabilities, which involves procedure modeling and property specification. In our work, we show the potential of our VerejoIn for analyzing Zigbee procedures. Another important lesson is that IoT security systems should be designed in a way that takes different layers (e.g., the cloud layer and the cyber layer) into consideration. As a result, when anomalies happen, a security system that collects evidence from multiple layers can better localize the root causes. For example, when the platform is informed of the offline state of the CO_2 sensor and the anomaly of related automation rules (e.g., the disabled alarm), the collected Zigbee network rejoin response (recording the PAN FULL state of parent devices) can help security analysts to identify the root cause of the anomaly, i.e., Zigbee capacity exploitation and offline attack.

Addressing our reported vulnerabilities. In order to fix our reported vulnerabilities, we collaborate with Zigbee Alliance and revise the currently released specifications (R21/R22) and the specification to be released (R23) [12]. We further update our model and initiate the model checking to verify whether these revisions are valid. As a result, no counterexamples are reported, which means

that these revisions successfully address our reported vulnerabilities. Descriptions of the revisions are as follows.

- *Improve the aging-out process.* The broken aging-out process allows phantom devices to permanently take up the parent device's child table. To clean up these phantom devices, the revision requires that *if an orphaned device does not send at least one network encrypted message within apsSecurityTimeOutperiod milliseconds, it shall be deleted from the child table.* Since phantom devices do not hold the network key and cannot send the encrypted message, they can only be temporally accepted onto the network.
- *Strengthen checks of device properties.* It has been shown that the child table can be manipulated by unsolicited devices. Therefore, the revised specification adds a new subsection which requires that *parent devices initiate strict checks before the table is updated by the insecure packet.* Specifically, the insecure packet must not overwrite legitimate data in the child table. As a result, the phantom device cannot tamper with legitimate devices' information stored in the child table, and cannot create inconsistent issues.
- *Remove the support of the well-known link key.* Since the specification still supports the global link key, we help the alliance to comprehensively remove it from the specification, and add a new section which regulates the rejoin behavior. Specifically, the parent device should specify a unique link key for each end device when it joins the network. When the parent device receives a rejoin request from an end device, it shall first determine whether the used link key is the global one. If that is the case, the rejoin should be rejected. Otherwise, the parent device will transmit the network key encrypted with the unique link key. The child device with the unique link key shall not accept any commands (e.g., the key transport command) encrypted with the default link key.

We further test vendors' patches to check if they follow the updated specification. Specifically, for each evaluated device (see Table 2 on Page 10), we first upgrade its firmware to the latest version. Then we use ZigHomer to initiate our PoC attacks again. The result shows that all vendors successfully patched their devices. (1) All phantom devices are cleaned up within seconds, and the state of full capacity is eliminated. (2) The spoofed rejoin request cannot tamper with the child table, and the parent device will not ask legitimate child devices to leave the network. Finally, (3) the parent device will not transmit network keys to any devices which use the well-known link key, and the offline child device will not accept the transmitted network key which is encrypted with the well-known link key.

Designing model checking tools. Modeling and verifying real-world Zigbee systems with diverse actors and operations is complicated. Our tool Verejoin makes an attempt and shows its effectiveness. Currently, it focuses on the rejoin procedure and supports several operation types. Extending the scope of supported operations/procedures will be our future research. Moreover, Improving Verejoin to support more advanced techniques (e.g., compositional verification [26, 44]) for analyzing larger numbers of actors is also a potential direction for future research.

8 RELATED WORK

Link key vulnerability. Security analysts have been aware that the transmission of network keys (encrypted with well-known link keys) is not secure. Specifically, prior work [7, 20, 24, 36, 48, 52, 56] focuses on how to trigger the network key transmission and cause the network key leakage. [7, 24, 36, 52, 56] target Zigbee's join procedure. They show that attackers can trigger network key transmission by social engineering and jamming attacks, which induce users to initiate the join procedure and add devices. However, since these attacks require continuous jamming (e.g., jamming the endless beacon) and users' involvement (the join procedure can be only initiated by users), they are difficult to initiate in the real world. [20, 48] target Zigbee's rejoin procedure, and suggest that attackers can impersonate a legitimate end device to send spoofed rejoin requests, which trigger the network key transmission. However, they require attackers to have the legitimate end device's property information (e.g., the address) and did not provide instructions on how to get the information.

Compared with the prior work, our work constructs easily deployable attacks to cause network key leakage. Specifically, our Network Key Leakage Attack does not require users' involvement and any device information, such that attackers can easily initiate it in the real world. Besides leaking the network key, our work also extends Zigbee's attack vector and constructs a new attack (i.e., the Hijacking Attack) that replaces the legitimate network key.

Attacks on ZigBee. Prior work focuses on the design and implementation flaws in Zigbee application profiles [40, 41, 47]. Specifically, the implementation of ZLL profiles on Amber devices is vulnerable, which provides an attack surface for attackers to reset and hijack the Philips light bulb [47]. Design flaws in ZLL have also been identified and can be exploited to take control of ZLL-based lighting systems [40]. Besides the ZLL profile, ZigBee 3.0 is discovered to be insecure by design, and attackers can exploit the Touchlink commissioning mechanism specified by Zigbee 3.0 [41]. In contrast to these previous works on design flaws in specific application profiles, our study considers the network rejoin procedure, which is common to all Zigbee devices and is supported by most application profiles. Furthermore, the impact of our reported vulnerabilities ranges from the denial of service to device hijacking, which is more severe.

Zigbee devices are also vulnerable to jamming attacks [46], which jams the communication bandwidth and disables devices. We highlight that our proposed denial-of-service attacks (Capacity Exploitation and Offline Attack) have less restrictive assumptions and achieve more damaging consequences. Specifically, our attack can be initiated within seconds, and the denial of service can last for hours (even permanently). The jamming attack, however, requires consistent traffic monitoring and jamming, which is infeasible in the real world. Moreover, our attacks exploit device properties (e.g., the capacity and the child table), while the jamming attack does not affect device properties.

Model-based vulnerability discovery. Prior work uses fuzzing, symbolic execution, and formal verification techniques to discover vulnerabilities in diverse protocols [18, 23, 31, 54, 55]. Specifically, a model-based approach is proposed to discover vulnerabilities in TCP congestion control automatically [31], and another approach is

proposed to identify new forms of idle port scan attacks using model checking techniques [23]. Researchers recently model the delegation process for IoT platforms and confirm several design flaws [55]. In our work, we design a tool VEREJOIN to model and analyze the network rejoin procedure in the Zigbee protocol. Consequently, our tool discovers widespread vulnerabilities in the procedure.

Appified platform security. There are plenty of works focusing on the security of automation rules [16, 17, 17, 21, 29, 51]. For example, rule interactions through the physical channels (e.g., temperature) may introduce unexpected behaviors of devices [21]. Installed automation rules can be chained together to bring in new threats [17, 29, 51]. Our evaluation results and discussions show that our reported vulnerabilities can also affect these automation rules. Even worse, existing security systems [16, 17] cannot address these vulnerabilities.

9 CONCLUSIONS

Zigbee is a dominant IoT communication protocol and the large number of deployed Zigbee devices highlights the importance of Zigbee's security. In this work, we first identify a set of security properties for IoT devices, then we use the model checking technique and design a Zigbee security checking tool VEREJOIN. Using our tool, we successfully report three design flaws and construct four PoC attacks, which are all acknowledged by Zigbee Alliance and the affected vendors. We also design a Zigbee testing tool ZIGHOMER and implement our constructed PoC attacks on it. We show that ZIGHOMER outperforms existing tools, and it is also flexible such that users can specify their own testing tasks. Finally, we use ZIGHOMER to evaluate our reported vulnerabilities in terms of prevalence and severity. The result shows that our reported vulnerabilities can cause huge threats to existing Zigbee devices and automation rules provided by IoT platforms. Further discussions highlight the risk of device procedures that trade security for utility, and the importance of designing comprehensive IoT security systems. Finally, we collaborate with Zigbee Alliance and successfully amend the current and future versions of the Zigbee specification.

ACKNOWLEDGMENTS

The work of John C.S. Lui is supported in part by the RGC R4032-18. We would also like to express our gratitude to our shepherd Dr. Omar Alrawi and anonymous reviewers for their constructive comments. Thanks to Zigbee Alliance and the device vendors who responded to our vulnerability report.

REFERENCES

[1] [n.d.]. *AN1233: Zigbee Security.* https://www.silabs.com/documents/public/application-notes/an1233-zigbee-security.pdf
[2] [n.d.]. *What's New in Zigbee 3.0.* https://www.ti.com/lit/an/swra615a/swra615a.pdf
[3] [n.d.]. *ZigBee 3.0 – Facilitating the Internet of Things.* https://www.nxp.com/docs/en/brochure/75017677.pdf
[4] 2018. Hub tells ZigBee End Device to Leave on Rejoin. https://community.smartthings.com/t/hub-tells-zigbee-end-device-to-leave-on-rejoin/128646.
[5] 2020. AN1233: Zigbee Security. https://www.silabs.com/documents/public/application-notes/an1233-zigbee-security.pdf.
[6] 2020. Overview of Samsung SmartThings Secure Mode. https://www.samsung.com/sg/support/mobile-devices/overview-of-samsung-smartthings-secure-mode/.
[7] Dimitrios-Georgios Akestoridis, Madhumitha Harishankar, Michael Weber, and Patrick Tague. 2020. Zigator: analyzing the security of zigbee-enabled smart homes. In *Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks*. 77–88.
[8] ZigBee Alliance. 2015. *ZigBee Specification.* https://zigbeealliance.org/wp-content/uploads/2019/11/docs-05-3474-21-0csg-zigbee-specification.pdf
[9] Zigbee Alliance. 2021. *Analysts Confirm Half a Billion Zigbee Chipsets Sold, Igniting IoT Innovation; Figures to Reach 3.8 Billion by 2023.* https://zigbeealliance.org/news_and_articles/analysts-confirm-half-a-billion-zigbee-chipsets-sold-igniting-iot-innovation-figures-to-reach-3-8-billion-by-2023/
[10] Zigbee Alliance. 2021. *Products Certified by ZigBee Alliance.* https://zigbeealliance.org/product_type/certified_product/
[11] Anonymous. 2021. *Rejoin Vulnerability: Attack videos and traffic logs.* https://sites.google.com/view/rejoin-vulnerability/attack-materials?authuser=0
[12] Anouymous. 2021. Responses from the alliance and vendors. https://sites.google.com/view/rejoin-vulnerability/alliancevendor-responses.
[13] IEEE Standards Association et al. 2011. 802.15. 4-2011 IEEE Standard for Local and Metropolitan Area Networks-Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs). *IEEE Standards Association* (2011).
[14] David Basin, Jannik Dreier, Lucca Hirschi, Saša Radomirovic, Ralf Sasse, and Vincent Stettler. 2018. A formal analysis of 5G authentication. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*. 1383–1396.
[15] Roberto Cavada, Alessandro Cimatti, Michele Dorigatti, Alberto Griggio, Alessandro Mariotti, Andrea Micheli, Sergio Mover, Marco Roveri, and Stefano Tonetta. 2014. The nuXmv symbolic model checker. In *International Conference on Computer Aided Verification*. Springer, 334–342.
[16] 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.
[17] 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.
[18] Chia Yuan Cho, Domagoj Babic, Pongsin Poosankam, Kevin Zhijie Chen, Edward XueJun Wu, and Dawn Song. 2011. MACE: Model-inference-Assisted Concolic Exploration for Protocol and Vulnerability Discovery.. In *USENIX Security Symposium*, Vol. 139.
[19] Edmund M Clarke Jr, Orna Grumberg, Daniel Kroening, Doron Peled, and Helmut Veith. 2018. *Model checking.* MIT press.
[20] dattatray. 2017. IoT Security – Part 6 (ZigBee Security - 101). https://payatu.com/blog/dattatray/zigbee-security-101.
[21] Wenbo Ding and Hongxin Hu. 2018. On the safety of iot device physical interaction control. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 832–846.
[22] Danny Dolev and Andrew Yao. 1983. On the security of public key protocols. *IEEE Transactions on information theory* 29, 2 (1983), 198–208.
[23] Roya Ensafi, Jong Chun Park, Deepak Kapur, and Jedidiah R Crandall. 2010. Idle Port Scanning and Non-interference Analysis of Network Protocol Stacks Using Model Checking.. In *USENIX Security Symposium*. 257–272.
[24] Xueqi Fan, Fransisca Susan, William Long, and Shangyan Li. 2017. Security analysis of zigbee. *MWR InfoSecurity* (2017), 1–18.
[25] Rob Gerth, Doron Peled, Moshe Y Vardi, and Pierre Wolper. 1995. Simple on-the-fly automatic verification of linear temporal logic. In *International Conference on Protocol Specification, Testing and Verification*. Springer, 3–18.
[26] Mihaela Gheorghiu, Dimitra Giannakopoulou, and Corina S Păsăreanu. 2007. Refining interface alphabets for compositional verification. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 292–307.
[27] Qi Hardware. 2013. *Ben-WPAN Overview.* http://downloads.qi-hardware.com/people/werner/wpan/web/
[28] Gerard J. Holzmann. 1997. The model checker SPIN. *IEEE Transactions on software engineering* 23, 5 (1997), 279–295.
[29] Kai-Hsiang Hsu, Yu-Hsi Chiang, and Hsu-Chun Hsiao. 2019. Safechain: Securing trigger-action programming from attack chains. *IEEE Transactions on Information Forensics and Security* 14, 10 (2019), 2607–2622.
[30] Syed Rafiul Hussain, Mitziu Echeverria, Imtiaz Karim, Omar Chowdhury, and Elisa Bertino. 2019. 5GReasoner: A property-directed security and privacy analysis framework for 5G cellular network protocol. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 669–684.
[31] Samuel Jero, Md Endadul Hoque, David R Choffnes, Alan Mislove, and Cristina Nita-Rotaru. 2018. Automated Attack Discovery in TCP Congestion Control Using a Model-guided Approach.. In *NDSS*.
[32] Constantinos Kolias, Georgios Kambourakis, Angelos Stavrou, and Jeffrey Voas. 2017. DDoS in the IoT: Mirai and other botnets. *Computer* 50, 7 (2017), 80–84.
[33] SILLICON LABS. 2021. Various keys used in ZLL Commissioning Plug-in in EmberZNet PRO stack. https://community.silabs.com/s/article/various-keys-used-in-zll-commissioning-plug-in-in-emberznet-pro-stack?language=en_US.
[34] Silicon Labs. 2021. *Zigbee 3.0 Prevention of Trust Center Rejoins best practices.* https://community.silabs.com/s/article/zigbee-3-0-prevention-of-trust-

center-rejoins-best-practices
[35] Leslie Lamport. 1977. Proving the correctness of multiprocess programs. *IEEE transactions on software engineering* 2 (1977), 125–143.
[36] Li Li, Proyash Podder, and Endadul Hoque. 2020. A formal security analysis of ZigBee (1.0 and 3.0). In *Proceedings of the 7th Symposium on Hot Topics in the Science of Security*. 1–11.
[37] Franco Loi, Arunan Sivanathan, Hassan Habibi Gharakheili, Adam Radford, and Vijay Sivaraman. 2017. Systematically evaluating security and privacy for consumer IoT devices. In *Proceedings of the 2017 Workshop on Internet of Things Security and Privacy*. 1–6.
[38] Francesca Meneghello, Matteo Calore, Daniel Zucchetto, Michele Polese, and Andrea Zanella. 2019. IoT: Internet of threats? A survey of practical security vulnerabilities in real IoT devices. *IEEE Internet of Things Journal* 6, 5 (2019), 8182–8201.
[39] Erich Mikk, Yassine Lakhnech, Michael Siegel, and Gerard J Holzmann. 1998. Implementing statecharts in PROMELA/SPIN. In *Proceedings. 2nd IEEE Workshop on Industrial Strength Formal Specification Techniques*. IEEE, 90–101.
[40] Philipp Morgner, Stephan Mattejat, and Zinaida Benenson. 2016. All your bulbs are belong to us: Investigating the current state of security in connected lighting systems. *arXiv preprint arXiv:1608.03732* (2016).
[41] Philipp Morgner, Stephan Mattejat, Zinaida Benenson, Christian Müller, and Frederik Armknecht. 2017. Insecure to the touch: attacking ZigBee 3.0 via touchlink commissioning. In *Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks*. 230–240.
[42] Madanlal Musuvathi, Dawson R Engler, et al. 2004. Model Checking Large Network Protocol Implementations.. In *NSDI*, Vol. 4. 12–12.
[43] Susan Owicki and Leslie Lamport. 1982. Proving liveness properties of concurrent programs. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 4, 3 (1982), 455–495.
[44] Corina S Păsăreanu and Dimitra Giannakopoulou. 2006. Towards a compositional SPIN. In *International SPIN Workshop on Model Checking of Software*. Springer, 234–251.
[45] Peter. 2016. *Extract Philips Hue's customized keys through reverse engineering*. https://peeveeone.com/?p=166
[46] Hossein Pirayesh, Pedram Kheirkhah Sangdeh, and Huacheng Zeng. 2020. Securing ZigBee communications against constant jamming attack using neural network. *IEEE Internet of Things Journal* 8, 6 (2020), 4957–4968.
[47] Eyal Ronen, Adi Shamir, Achi-Or Weingarten, and Colin O'Flynn. 2017. IoT goes nuclear: Creating a ZigBee chain reaction. In *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 195–212.
[48] Vishruta Rudresh. 2017. ZIGBEE SECURITY: BASICS (PART 3). https://research.kudelskisecurity.com/2017/11/21/zigbee-security-basics-part-3/.
[49] Rohit Sinha, Sriram Rajamani, Sanjit Seshia, and Kapil Vaswani. 2015. Moat: Verifying confidentiality of enclave programs. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. 1169–1184.
[50] SmartThings. [n.d.]. *SmartThings Public GitHub Repo*. https://github.com/SmartThingsCommunity/SmartThingsPublic
[51] Qi Wang, Pubali Datta, Wei Yang, Si Liu, Adam Bates, and Carl A Gunter. 2019. Charting the attack surface of trigger-action IoT platforms. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 1439–1453.
[52] Weicheng Wang, Fabrizio Cicala, Syed Rafiul Hussain, Elisa Bertino, and Ninghui Li. 2020. Analyzing the attack landscape of Zigbee-enabled IoT systems and reinstating users' privacy. In *Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks*. 133–143.
[53] Joshua Wright. 2009. Killerbee: practical zigbee exploitation framework. In *11th ToorCon conference, San Diego*, Vol. 67.
[54] Yan Xiong, Cheng Su, Wenchao Huang, Fuyou Miao, Wansen Wang, and Hengyi Ouyang. 2020. Smartverif: Push the limit of automation capability of verifying security protocols by dynamic strategies. In *29th {USENIX} Security Symposium ({USENIX} Security 20)*. 253–270.
[55] Bin Yuan, Yan Jia, Luyi Xing, Dongfang Zhao, XiaoFeng Wang, and Yuqing Zhang. 2020. Shattered chain of trust: Understanding security risks in cross-cloud iot access delegation. In *29th {USENIX} Security Symposium ({USENIX} Security 20)*. 1183–1200.
[56] Tobias Zillner and Sebastian Strobl. 2015. ZigBee Exploited–The good, the bad and the ugly. *Black Hat–2015. Available online: https://www. blackhat. com/docs/us-15/materials/us-15-Zillner-ZigBee-Exploited-The-Good-The-Bad-And-The-Ugly. pdf (accessed on 21 March 2018)* (2015).

A DATA AND OPERATIONS

Verejoin specifies a set of data that actors hold according to the Zigbee specification (Table 4). Note that actors of different types may have different data. For example, the network status captures different phases of end devices during the rejoin procedure, and it is only associated with the end device. Moreover, Verejoin supports four basic types of operations, each of which corresponds to the transmission of a specific Zigbee command. We construct these operations based on the command description in the Zigbee specification, and implement them using the Promela language. Note that one basic type of operation may have a few sub-types (See Section 3). We here use the abbreviated name of device properties (see Table 4) for a better illustration, and provide the definition of these operations as follows.

beaconRequest. An end device a_i can send a Beacon Request command to a parent device a_j. The operation $beaconRequest(a_i, a_j)$ is defined as:

$$\begin{cases} \texttt{Check}(P_{a_i}[\text{NS}] == \text{“Orphan”}) \\ \texttt{Notice}(a_i, a_j, \text{BEACON-REQUEST}) \\ P_{a_i}[\text{NS}] = \text{“Beacon Sent”} \end{cases}$$

Note that the $\texttt{Check}(p)$ primitive, where p is a logical function, will check whether p is true. If p holds, actors can proceed with the operation; otherwise actors have to abandon this operation. The $\texttt{Notice}(a_i, a_j, cmd)$ primitive, where cmd is a command id, denotes that a_i informs a_j of a command message with id equal to cmd.

beaconResponse: A parent device a_j sends a Beacon Response command to an end device a_i. Depending on whether a_i has an available capacity, this operation has two sub-types. Formally, when a_i has an available capacity, the operation $beaconResponse(a_i, a_j)$ as:

$$\begin{cases} \texttt{Check}(\texttt{Received}(a_i, a_j, \text{BEACON-REQUEST})) \\ \texttt{Check}(P_{a_j}[\text{CA}] > 0) \\ P_{a_i}[\text{NS}] = \text{“Rejoin Ready”} \\ P_{a_i}[\text{PA}] = P_{a_j}[\text{EA}] \end{cases}$$

Note that the $\texttt{Received}(a_i, a_j, cmd)$ primitive is a logical function which denotes whether a_i has been informed by a_j about a command message with id equal to cmd.

The operation $fullBeaconResponse(a_i, a_j)$ is also defined as follows when the capacity has been depleted.

$$\begin{cases} \texttt{Check}(\texttt{Received}(a_i, a_j, \text{BEACON-REQUEST})) \\ \texttt{Check}(P_{a_j}[\text{CA}] == 0) \\ P_{a_i}[\text{NS}] = \text{“Orphan”} \end{cases}$$

rejoinRequest: An end device a_i sends a Rejoin Request command to a parent device a_j. Depending on the security level of the request, this operation has two sub-types. Formally, when the request is encrypted using the network key (secure rejoin procedure), the operation $rejoinRequest(a_i, a_j)$ is defined as:

$$\begin{cases} \texttt{Check}(P_{a_i}[\text{NK}] == P_{a_j}[\text{NK}]) \\ \texttt{Check}(P_{a_i}[\text{NS}] == \text{“Rejoin Ready”}) \\ \texttt{Notice}(a_i, a_j, \text{REJOIN-REQUEST1}) \\ P_{a_i}[\text{NS}] = \text{“Rejoin Sent”} \end{cases}$$

When the request is in plaintext (trust center rejoin procedure), the operation $trustRejoinRequest(a_i, a_j)$ is defined as:

$$\begin{cases} \texttt{Check}(P_{a_i}[\text{LK}] == P_{a_j}[\text{LK}]) \\ \texttt{Check}(P_{a_i}[\text{NS}] == \text{“Rejoin Ready”}) \\ \texttt{Notice}(a_i, a_j, \text{REJOIN-REQUEST2}) \\ P_{a_i}[\text{NS}] = \text{“Rejoin Sent”} \end{cases}$$

rejoinResponse: A parent device a_i sends a Rejoin Response command to an end device a_j. Depending on the security level of the

Table 4: The list of data that actors hold

Actor Type	Data Name (Abbrev)	Data Type	Descriptions
Commons	Device Class (DC)	A	The type and legitimacy of the device
	Extended Address (EA)	P	The extended address of the device
	Link Key (LK)	S	The stored link key
	Network Key (NK)	S	The stored network key
End device	Receiver On When Idle (RO)	P	The *receiver on when idle* property
	Network Status (NS)	N	The network status of the end device
	Parent Address (PA)	P	The address of the connected parent device.
Parent device	Capacity (CA)	P	The available capacity to accommodate end devices
	Child Table (CT)	P	A table of data copies for connected end devices

In the "Data Type" column, (P, N, A, S) represents (*device property, network status, auxiliary data, security information*) respectively.

Table 5: Examples of affected Smart Apps

App Name	Class	Functionalities	Descriptions of Impacts
Cameras On When I'm Away	a	The camera will be turned on when no one is at home.	The invalidation of *presence sensors* prevents the detection of persons, and freezes the camera operations.
Flood Alert	b	Get a push notification or text message when water is detected where it does not belong.	The invalidation of *flood sensors* prevents users from getting notifications about potential floods.
Weather Underground PWS Connect	c	Sense local temperature/humidity and upload the data to the weather station.	The invalidation of *temperature/humidity sensors* results in the upload of meaningless data.
Good Night	d	Changes mode when motion ceases.	The invalidation of motion sensors prevents the home mode from transitions.

In the "Class" column, (a, b, c, d) represents (*Device functionality interference, Silent notification, Web service interference, Mode interference*) respectively (See Section 6.3).

response, this operation has two sub-types. Formally, if the response is encrypted using the network key (secure rejoin procedure), the operation $rejoinResponse(a_i, a_j)$ is defined as:

$$\begin{cases} \texttt{Check}(\texttt{Received}(a_i, a_j, \text{REJOIN-REQUEST1})) \\ P_{a_j}[\text{NS}] = \begin{cases} \text{“Online”}, P_{a_i}[\text{CA}] > 0 \\ \text{“Orphan”}, P_{a_i}[\text{CA}] = 0 \end{cases} \\ P_{a_i}[\text{CA}] = \begin{cases} P_{a_i}[\text{CA}] - 1, P_{a_i}[\text{CA}] > 0 \\ 0, P_{a_i}[\text{CA}] = 0 \end{cases} \\ \texttt{Update}(P_{a_i}[\text{CT}], P_{a_j}[\text{EA}], P_{a_j}[\text{RO}]) \end{cases}$$

Note that the $\texttt{Update}(\text{CT}, \text{EA}', \text{RO}')$ primitive is designed for the update of the child table CT, in which each entry is in the form of (EA, RO). Specifically, if the address EA′ is already in CT, this primitive will update the RO value of the corresponding entry using RO′; otherwise this primitive will create a new entry (EA′, RO′) and store it into CT.

When the response is in plaintext (trust center rejoin procedure), the operation $trustRejoinResponse2(a_i, a_j)$ is defined as:

$$\begin{cases} \texttt{Check}(\texttt{Received}(a_i, a_j, \text{REJOIN-REQUEST2})) \\ P_{a_j}[\text{NS}] = \begin{cases} \text{“Online”}, P_{a_i}[\text{CA}] > 0 \\ \text{“Orphan”}, P_{a_i}[\text{CA}] = 0 \end{cases} \\ P_{a_i}[\text{CA}] = \begin{cases} P_{a_i}[\text{CA}] - 1, P_{a_i}[\text{CA}] > 0 \\ 0, P_{a_i}[\text{CA}] = 0 \end{cases} \\ \texttt{Update}(P_{a_i}[\text{CT}], P_{a_j}[\text{EA}], P_{a_j}[\text{RO}]) \\ P_{a_j}[\text{NK}] = P_{a_i}[\text{NK}] \end{cases}$$

Additional notes. We need to point out that some of the above formalisms may not strictly follow the command description in the Zigbee specification. Specifically, we highlight the following two adjustments we made as follows.

- According to the Zigbee specification, the Beacon Request command is transmitted in a broadcast mode. However, we model it as a unicast command because Promela does not provide corresponding grammar support. To comply with the specification, we require that when an end device is going to send the Beacon Request command, it will send the command to all parent devices.
- According to the Zigbee specification, the unencrypted Rejoin Response command does not transmit the network key, but the Zigbee command Key Transport command is responsible for the key transmission. Because the Key Transport

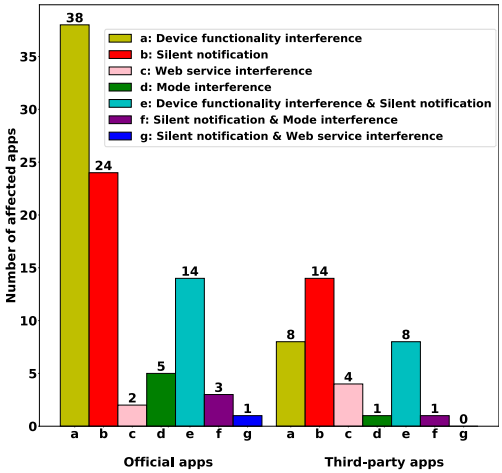


Figure 9: Affected functionalities of automation rules

command always follows by the unencrypted Rejoin Response command, we merged the functionality of these two commands, and give the formalism of the *trustRejoinResponse* operation.

B EXAMPLES OF AFFECTED AUTOMATION RULES

We list four example rules which are affected by our reported attacks in Table 5. Specifically, each of these rules is from the SmartThings' official or third-party repository. All these examples are classic rules because each of them performs a specific basic functionality as introduced in Section 6.3. Here, we introduce their functionalities and details of the impact on these rules. The result implies that if users heavily rely on these automation rules, our reported attacks can cause significant threats.