ZCOVER: Uncovering Z-Wave Controller Vulnerabilities Through Systematic Security Analysis of Application Layer Implementation

Nkuba Kayembe Carlos*, Jimin Kang*, Seunghoon Woo*†, and Heejo Lee*†
*Department of Computer Science and Engineering, Korea University
Email: {carlosnkuba, ziminii, seunghoonwoo, heejo}@korea.ac.kr

***Abstract*—The increasing use of smart home technologies has raised concerns about security vulnerabilities, particularly in Z-Wave systems. Existing approaches hold the promise of assessing Z-Wave security in slave devices but fall short of being effectively applied to discover vulnerabilities in Z-Wave controllers, which are central to Z-Wave systems. We present ZCOVER, a framework for systematically analyzing the application layer of Z-Wave controllers to uncover security vulnerabilities. By extracting the known and unknown properties of the Z-Wave controller and utilizing mutation that considers the correlations of the Z-Wave packet frame fields, ZCOVER can effectively discover unknown vulnerabilities in the target Z-Wave controller. Evaluation on nine real-world Z-Wave devices showed that ZCOVER outperformed existing Z-Wave security research, by discovering 15 previously unknown critical vulnerabilities with 12 new CVE IDs assigned. ZCOVER can be utilized as a resource for ensuring the security of Z-Wave controllers in building a secure Z-Wave smart home.**

***Index Terms*—Internet of Things, Z-Wave, Smart home, Security analysis, Fuzzing, Vulnerability discovery.**

I. INTRODUCTION

The use of Z-Wave [1] smart home technologies has ushered in a new era of convenience and connectivity [2], allowing users to monitor and control various aspects of their living spaces. However, this increased connectivity brings new security challenges, as smart home devices become targets for malicious actors exploiting vulnerabilities [3].

While smart home device security (*e.g.*, cameras and smart locks) has been widely studied [4], the *controller* has often been overlooked. As the central hub in Z-Wave systems, the controller manages device communication between diverse slave devices. Despite its critical role, its security remains underexplored and potentially vulnerable. Despite its pivotal role, the security of the Z-Wave controller has received limited scrutiny, leaving it potentially vulnerable.

This gap in research highlights the need for a systematic security analysis framework tailored specifically to the application layer implementation of Z-Wave controllers, aiming to uncover potential vulnerabilities and enhance the overall security posture of Z-Wave smart home systems.

Several existing approaches have attempted to test the security and privacy assurance of the Z-Wave protocol and devices. Unfortunately, they (1) are not suitable for analyzing the security of the Z-Wave controller (*e.g.*, [5]), (2) do not focus on examining the main controller's application layer security (*e.g.*, [6]–[8]), and (3) fail to sufficiently consider the proprietary properties of the controller (*e.g.*, [9]), resulting in low efficiency in identifying vulnerabilities. For these reasons, existing approaches are not effective at detecting unknown vulnerabilities in the Z-Wave controller.

†Corresponding authors: Seunghoon Woo and Heejo Lee

Challenges. We face several challenges in assessing the Z-Wave controller to discover unknown vulnerabilities.

First, because of the closed nature of Z-Wave implementation, the lack of accessibility to devices' source code hampers our ability to perform a thorough code review and identify potential vulnerabilities from the source. Therefore, we have to assess the device based on black-box testing (*e.g.*, fuzzing). However, it is difficult to obtain detailed information on the core properties (*e.g.*, command classes) of controllers for efficient mutation, because proprietary documentation is often incomplete or limited, making it difficult to understand the operation of the controller. Moreover, the mutation strategy should consider the Z-Wave application layer implementation within the controller, necessitating precise and sophisticated techniques to generate meaningful test cases.

Our approach. To overcome these challenges, we propose a new approach called **ZCOVER (Z-Wave COntroller Vulnerability discovERy)**, a security analysis framework that helps find systematic information on a target Z-Wave controller to discover previously unknown vulnerabilities.

The main ideas of ZCOVER, which are significantly different from existing approaches, are (1) comprehensively extracting both the known and unknown properties (*e.g.*, command classes) of the Z-Wave controller and (2) performing fuzzing using position-sensitive mutation by considering the correlation of Z-Wave packet fields.

By identifying both the known (listed) and unknown (unlisted) properties of the target controller, ZCOVER can assess the target controller more effectively. By performing mutations based on the controller's properties and considering the correlation between Z-Wave packet fields, ZCOVER increases the possibility of detecting potential vulnerabilities from the target controller.

Previous approaches did not consider unknown or proprietary command classes (`CMDCL`s) for efficient fuzzing of the Z-Wave controller, nor did they effectively consider the correlation of the Z-Wave application layer frame fields. However, we addressed these issues by fingerprinting a target controller for known `CMDCL`s and conducting a mutation strategy based on the discovered `CMDCL`s definition.

Specifically, ZCOVER comprises the following three phases: (1) known properties fingerprinting (Section III-B), (2) un-

2158-3927/25/$31.00 ©2025 IEEE
DOI 10.1109/DSN64029.2025.00068

known properties discovery (Section III-C), and (3) position-sensitive mutation (Section III-D).

Efficient fuzzing requires understanding the controller's properties. ZCOVER achieves this by combining new *passive* and *active* scannings to extract known and unknown properties. Through passive scanning, ZCOVER captures properties such as home ID and CMDCLs by intercepting packet exchanges in a normal Z-Wave network. Active scanning further identifies these properties using the Z-Wave node information frame (NIF) requests. To uncover hidden CMDCLs, ZCOVER leverages the Z-Wave specification and validation tests to identify unlisted but supported properties.

Using the identified properties, ZCOVER generates test packets that are rarely rejected and effectively reveal vulnerabilities. Its position-sensitive mutation, which considers packet field correlations based on CMDCL, improves the fuzzing efficiency compared to simple mutation approaches.

When we applied ZCOVER to nine real-world Z-Wave devices, it identified **15 critical zero-day vulnerabilities**, from which **12** were assigned **new CVE IDs** (see Section IV-A). All detected vulnerabilities were reported to the respective vendors and were all confirmed. Furthermore, our experiments comparing ZCOVER with a recent Z-Wave vulnerability discovery approach (*i.e.*, VFUZZ [9]) demonstrated that ZCOVER is faster and more effective at identifying unknown vulnerabilities in Z-Wave controllers. Unlike VFUZZ, which focuses on slave devices, ZCOVER looks at the correctness of controller implementations' handling of application layer payload. (see Section IV-C).

Contributions. We summarize our contributions below.

- We propose ZCOVER, a new approach for discovering potential security issues in a Z-Wave controller. The key technical contributions are a methodology for identifying known and unknown properties of the Z-Wave controller and a mutation technique that considers the correlation of Z-Wave packet fields.
- We have demonstrated that vulnerabilities can be exploited in Z-Wave controllers known to use encryption for security (*i.e.*, Security 2 encapsulation). ZCOVER can address these concerns in advance, and we suggest attack remediation measures for prevention.
- ZCOVER discovered 15 zero-day vulnerabilities (with 12 new CVE IDs assigned) in nine real-world Z-Wave devices. All vulnerabilities were confirmed by the corresponding vendors, and our findings will be reflected in the upcoming 2024 Z-Wave specification release. We provide sample videos in [10], [11] showcasing found vulnerabilities impact on real Z-Wave devices.

II. BACKGROUND, THREAT MODEL AND CHALLENGES

In this section, we first present the background knowledge of the Z-Wave protocol, including its architectural components and security mechanisms. We then outline the threat model relevant to IoT systems in smart homes, along with the challenges to address Z-Wave threats.

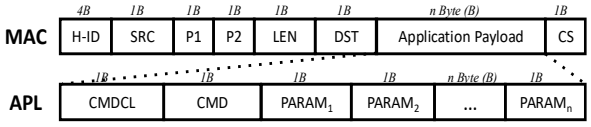


Fig. 1: Z-Wave basic frame structure details.

A. Z-Wave protocol overview

Z-Wave is a wireless communication protocol designed for home automation and smart home applications. It is a low bandwidth, low-power mesh protocol that operates in the sub-gigahertz frequency range (800 - 900 MHz) and does not interfere with other household electronics using Bluetooth, Wi-Fi, and ZigBee in the 2.4 GHz range. Z-Wave devices typically include sensors, actuators, and controllers, all interconnected within a mesh network topology. Each device from different vendors is equipped with a Z-Wave chipset (*e.g.*, 100 to 800 series) to ensure interoperability. In addition, the Z-Wave application layer (APL) defines various command classes and profiles that facilitate communication and interoperability between controllers and slave devices, allowing users to create customized smart home setups tailored to their specific needs [12], [13].

Figure 1 summarizes the structure of a Z-Wave frame. The maximum MAC frame size is 64 bytes. We briefly introduce the function of each packet field [9], [14], [15].

1) H-ID: Home ID value of a Z-Wave network.
2) SRC: The sender ID.
3) Frame control (P1 & P2): Specify the packet type (P1) and routing information (P2).
4) LEN: Packet length.
5) DST: The receiver ID.
6) CMDCL: Command class related to certain functionality of a device.
7) CMD: Command to be executed by the receiver device.
8) PARAM: Command payload values (*e.g.*, lock or unlock a smart door lock device).
9) CS: Checksum of a Z-Wave packet.

1) Encapsulation of Z-Wave data transmission: Z-Wave data is exchanged between controllers and devices using three transport modes [9], [12]:

- **No Security.** Data is sent without encryption, relying on basic checksums (CS-8/CRC-16). Legacy devices are vulnerable to injection attacks due to inadequate protection mechanisms.
- **Security 0 (S0).** Uses AES-128 encryption but is susceptible to MITM attacks due to a fixed temporary key during key exchange [7].
- **Security 2 (S2).** Employs ECDH for secure key derivation and AES-128-CMAC for integrity. Despite its enhancements, Z-Wave protocol still has flaws in its specification and implementation (see Section IV-A).

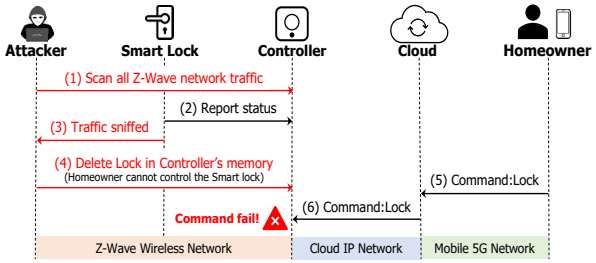


Fig. 2: The attacker, positioned approximately 70 meters away, uses a dongle connected to a device to scan, craft, and inject DoS traffic, compromising the controller's memory.

2) Z-Wave controller: The Z-Wave controller, also known as hub, serves as a gateway between Z-Wave wireless traffic and IP Internet traffic, facilitating communication with the cloud server (see Figure 2). The controller is also responsible for event reporting and automation activation. Any attack on the controller would render the entire smart home system non-functional, making the user unable to control devices through mobile apps or web service interfaces (see Section IV-A). In this paper, ZCOVER looks at the correctness of controller implementations' handling of Z-Wave application layer payload.

B. Threat model

The primary threat actors in the Z-Wave smart home systems include malicious insiders, external attackers, and unintentional threats such as device misconfigurations.

- **Malicious insiders** may seek to exploit their privileged access to the Z-Wave controller to compromise the integrity or confidentiality of the system.
- **External attackers** attempt to exploit vulnerabilities in the Z-Wave protocol or controller implementation to gain unauthorized access to sensitive information in memory or disrupt normal system operation.
- **Unintentional threats**, such as device malfunctions or misconfigurations, can also pose risks to the security and reliability of Z-Wave home systems, potentially leading to unintended consequences or system failures.

1) Attack scenario: Figure 2 illustrates a potential attack scenario within Z-Wave home automation systems. In this scenario, both the controller and the smart door lock support and utilize the latest S2 encrypted communication transport. An external attacker, positioned outside the house within a range of 10 to 70 meters, employs a hardware dongle connected to a laptop to scan and record all traffic from the Z-Wave controller and slave devices [7], [9], [16]. While S2 communication encrypts only the Z-Wave packet application layer, the attacker can still sniff remaining packet fields such as home ID, source, and destination details.

An attacker with knowledge of the Z-Wave network can exploit application layer *proprietary* `CMDCLs` and `CMDs` to create malicious *unencrypted* payloads that alter or delete the controller's internal memory information for controlling the smart door lock. These unencrypted malicious payloads can be wirelessly injected into the controller, causing it to erase the S2 door lock data. As a result, the homeowner will be unable to lock the smart door remotely or locally using their smartphone app, as the controller will no longer recognize the lock. Furthermore, memory tampering attacks on the controller hinder the homeowner from controlling any device in the network (see Section IV-A). These attacks occur without the homeowner's knowledge. To prevent such intrusions, security testing of the central controller in Z-Wave smart homes is essential.

C. Challenges for enhancing Z-Wave security

Improving Z-Wave security faces significant obstacles due to the proprietary and closed nature of Z-Wave implementations, making source code inaccessible. This hampers in-depth code reviews, leaving fuzzing as an effective alternative for discovering vulnerabilities (*e.g.*, [5], [9], [17]). However, two key challenges complicate fuzzing efforts.

Diversity in implementations. Z-Wave controllers only implement specific `CMDCLs` relevant to their functionality, not all specified `CMDCLs`. Fuzzing without targeting these used `CMDCLs` leads to most test packets being rejected, reducing efficiency. Identifying implemented `CMDCLs` in firmware is difficult due to restricted proprietary source code, which limits access to critical details for testing [6].

Designing effective mutations. The hierarchical structure of Z-Wave packets (`CMDCL`, `CMD`, and `PARAM`) varies by device type. Crafting semi-valid and invalid payloads requires a deep understanding of the protocol to uncover vulnerabilities. These payloads must be sophisticated enough to test exception and error conditions without being rejected by the controller's basic checks. Overcoming these challenges is crucial for uncovering new security flaws.

III. METHODOLOGY

In this section, we describe the design of ZCOVER, a comprehensive security analysis framework for Z-Wave controllers tailored to the application layer implementation.

A. Overview

ZCOVER basically performs fuzzing [18] on the target Z-Wave controller. During this process, it extracts and leverages useful properties of the controller and employs an efficient packet mutation algorithm. The key features of ZCOVER lie in (1) leveraging proprietary hidden properties (*i.e.*, hidden `CMDCLs` of the controller) along with the known listed properties of the controller and (2) performing packet mutations by considering the position of Z-Wave packet fields. With each of these new features, we can overcome the challenges introduced in Section II-C.

Figure 3 shows the high-level workflow of ZCOVER. ZCOVER employs three distinct approaches to uncover potential vulnerabilities: known properties fingerprinting, unknown properties discovery, and position-sensitive mutation.

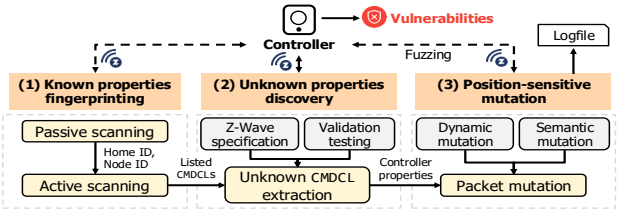


Fig. 3: High-level overview of ZCOVER.

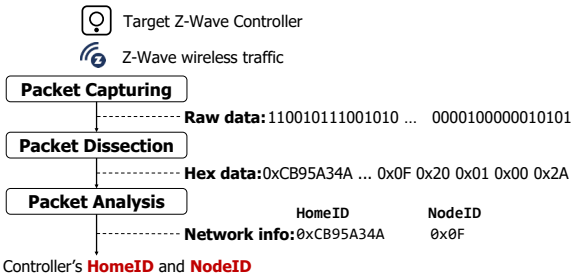


Fig. 4: Diagram of the passive scanning of ZCOVER.

1) **Known properties fingerprinting.** Initially, the *passive* scanner of ZCOVER sniffs Z-Wave traffic from a target controller to gather network information such as home ID and device node ID. Next, the *active* scanner conducts device reconnaissance to retrieve other known properties such as listed supported CMDCLs.
2) **Unknown properties discovery.** Upon obtaining the known properties, ZCOVER provides them to the unknown CMDCL extractor to uncover additional hidden or unlisted CMDCLs that the controller should implement according to its classification.
3) **Position-sensitive mutation.** The extracted properties, both known and unknown, are then sent to the mutator module, which mutates application payloads, considering CMDCL sub-categories command, parameter-position semantics, and syntax of the Z-Wave packets. ZCOVER conducts this process for each test case.

Design assumption. The goal of ZCOVER is to identify unknown vulnerabilities within the Z-Wave controller's application layer. We designed ZCOVER to run as an external, independent entity of the smart home due to the closed-source nature of the Z-Wave controller chipset, which does not allow the generation of mutated Z-Wave packets [5]. To demonstrate the practicality of ZCOVER, it does not have privileged access to the network and operates externally using specialized hardware to sniff, analyze, craft, and inject Z-Wave packets into the network.

B. Known properties fingerprinting

ZCOVER first identifies the controller's known properties using two techniques, namely *passive* and *active* scanning, which examine communications between the Z-Wave controller and known devices. ZCOVER targets three key properties: home ID, node ID, and listed (*i.e.*, known) supported CMDCLs, used by the Z-Wave controller to validate a message trustworthiness. Recognizing these is crucial, as ignoring them results in test packet rejection.

1) Passive scanner: First, ZCOVER identifies the home ID and node ID through passive scanning. ZCOVER starts in scanning mode to search for available Z-Wave networks. As soon as communication occurs between a slave device and the controller, ZCOVER captures the packet and retrieves the network home ID and the device node ID associated with the packet exchange. Figure 4 shows passive scanning of ZCOVER, which encompasses the following three steps.

1) **Packet capturing.** ZCOVER verifies that the Z-Wave transceiver dongle is configured with a valid radio frequency (RF) and sampling rate (*e.g.*, 868 or 908 MHz). ZCOVER then scans any Z-Wave wireless traffic, filters out the noise by removing specific repetitive bytes in the signal, and retrieves the raw binary data.
2) **Packet dissection.** ZCOVER converts the raw Z-Wave packet data into hexadecimal values for better representation and understanding.
3) **Packet analysis.** ZCOVER analyzes individual packet bytes to extract the Z-Wave network home ID and device node ID of the sender and receiver.

Even if the target controller utilizes S2 communication, the home and node IDs can be retrieved because S2 encrypts only the APL layer payloads. Therefore, ZCOVER identifies the position within the Z-Wave packet frame to obtain the network details (*e.g.*, bytes index zero to three for home ID; see Figure 1). After identifying the home and node IDs, ZCOVER passes them to the active scanner.

2) Active scanner: Based on the properties collected in the previous steps, ZCOVER requests additional details, *i.e.*, listed supported CMDCLs, from the target controller. Here, ZCOVER leverages a node information frame (NIF) request; when we request the controller via a NIF packet, the controller responds with its listed supported CMDCLs (*e.g.*, controller D4 listed only 17 CMDCLs; see Table IV). ZCOVER carefully verifies the CMDCLs of transmitted packets, because this reveals the controller's capabilities and functionalities.

Active scanning comprises the following three steps.

1) **Dynamic device interrogation.** ZCOVER initiates *active* communication with the controller on the network, sending targeted interrogation packets (*i.e.*, device state) to request responses.
2) **Listed property querying.** ZCOVER systematically queries the target controller for its supported known properties or CMDCLs (*i.e.*, via NIF packets).
3) **Response analysis.** Upon receiving device responses from the NIF request, ZCOVER analyzes the data to extract relevant information, including listed supported CMDCL to build an initial profile of the target controller.

The known properties, *i.e.*, home ID, node ID, and listed

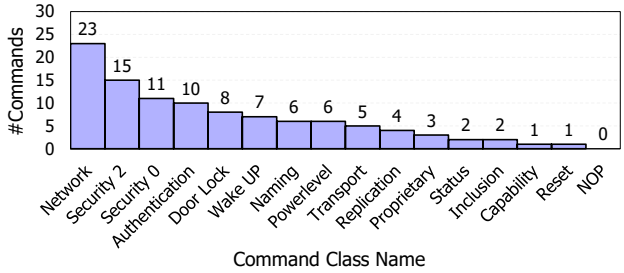


Fig. 5: Visualization of selected Z-Wave command classes (*i.e.*, we listed 15 CMDCLs for better visualization) and their corresponding commands distribution.

supported CMDCLs, identified in this section are used for effective fuzzing in the final phase (see Section IV-B1).

C. Unknown properties discovery

Next, ZCOVER identifies proprietary (i.e., hidden) CMDCLs of the target Z-Wave controller, which are often undocumented and known only to manufacturers under a non-disclosure agreement (NDA). If poorly implemented, these unlisted CMDCLs can be exploited by attackers.

To detect and utilize these hidden CMDCLs for fuzzing, ZCOVER (1) leverages the Z-Wave specifications and (2) performs a systematic validation testing. In particular, ZCOVER leverages the Z-Wave specification to identify unlisted CMDCLs related to the controller. Next, ZCOVER employs a systematic validation testing to verify whether the detected CMDCLs are functional and supported by the target controller.

1) Leveraging public Z-Wave specification: The Z-Wave Alliance maintains the Z-Wave specification [19], which, as of November 2024, lists 122 CMDCLs. It details CMDCLs structures, CMD types, command flows, and supported functionalities. Understanding this specification is crucial for identifying deviations and potential unlisted proprietary properties related to the controller.

Each CMDCL has multiple CMDs listed, along with an ID in hexadecimal format and whether the CMD is *controlling* (sent by a controller) or *supporting* (sent by a slave device in response). The CMDs can be categorized into different types, *e.g.*, *Get* to retrieve information from a device and *Set* to configure or control a device.

Locating missing CMDCLs. Identifying missing CMDCLs requires understanding that supported CMDCLs vary by Z-Wave device type. To analyze this, ZCOVER references the Z-Wave specification and an XML file listing Z-Wave application layer CMDCL definitions [20]. Using an automated script, ZCOVER parses these sources and clusters CMDCLs that a controller should support.

A Z-Wave controller is expected to support CMDCLs related to application functionality, transport encapsulation, management, and networking. For instance, CMDCLs such as Transport, Security 0, and Security 2 are inherently associated with the controller. By clustering CMDCLs based on function, fuzzing efforts can focus on specific controller-managed functionalities, ensuring targeted testing in critical areas where vulnerabilities are most likely.

The clustered CMDCLs serve as a baseline to detect CMDCLs that should be implemented but were not revealed during fingerprinting. This method helps pinpoint proprietary CMDCL candidates. Using this, ZCOVER extracts all unlisted CMDCLs along with their associated Commands (CMDs) and Parameters (PARAMs) that a target controller should support. Among 122 CMDCLs listed in the specification, ZCOVER inferred **26** unlisted CMDCLs relevant to the controller. These 26 CMDCLs were absent during the known properties fingerprinting stage, where ZCOVER initially identified only 17 CMDCLs (see Section III-B).

All clustered CMDCLs are treated as valid when generating test packets for fuzzing. This approach enables ZCOVER to uncover additional hidden CMDCLs beyond those explicitly listed as supported, thereby improving the efficiency of vulnerability discovery.

Prioritizing CMDCLs. Upon examining the specification, first, we can set the priority of unlisted CMDCLs during fuzzing. CMDCL supports multiple CMDs, and the number of CMDs supported by each CMDCL varies (see Figure 5). Here, ZCOVER gives higher priority to discovered unlisted CMDCLs that support more CMDs when mutating test packets during fuzzing; this follows our intuition that the more functionalities included, the higher the likelihood of potential implementation bugs.

2) Systematic validation testing: To verify the controller's support for the discovered unknown or unlisted CMDCLs, ZCOVER performs systematic validation testing. This process follows a sequential approach, evaluating CMDCLs from 0x00 to the upper limit of the identified CMDCL list. Through this method, ZCOVER uncovered **two** additional proprietary CMDCLs (0x01 and 0x02) that were absent from the official Z-Wave specification [19].

Notably, CMDCL 0x01, a Z-Wave network management property, was not explicitly listed by developers, likely due to incomplete implementation. This omission introduces critical security risks, as evidenced by our findings in Table III, where seven crucial vulnerabilities were associated with CMDCL 0x01. The lack of implementation robustness in this command class further underscores the necessity of ZCOVER systematic approach in uncovering hidden security flaws within Z-Wave controllers. The process of validating unknown CMDCLs is as follows.

Creating test cases. First, ZCOVER creates test cases that systematically send packets for each element of the discovered unknown CMDCL list starting from CMDCL 0x00. Thereafter, ZCOVER sends these test cases to the target controller to observe its response.

Monitoring responses. ZCOVER records the controller's responses to each test case, focusing on those indicating successful processing of unknown (*i.e.*, unlisted) CMDCL not identified

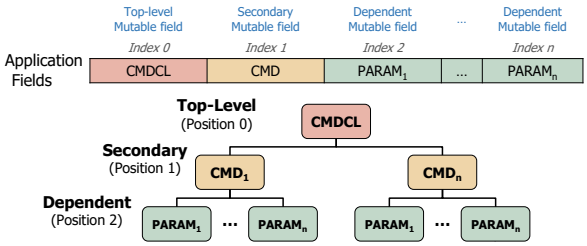


Fig. 6: Z-Wave application layer hierarchical structure.

during initial fingerprinting. CMDCLs not matching the initial report are flagged as valid proprietary candidates.

This task is effective in assessing the validity of the discovered hidden CMDCLs on the target controller. For instance, in our testbed, upon requesting the target controller D4 for its supported CMDCLs via a NIF frame request (*i.e.*, known properties fingerprinting), only **17** CMDCLs were listed by the controller (see Table IV). However, with above methods, we found an additional **28** unlisted CMDCLs candidates related to the controller to analyze.

By employing the aforementioned two techniques (*i.e.*, leveraging public specifications and systematic validation testing), we can uncover hidden CMDCLs for Z-Wave controllers. Thus, enhancing the understanding of the controller's full range of functionalities and potential CMDCLs to consider for vulnerabilities discovery during fuzz testing.

D. Position-sensitive mutation

Using the known and unknown properties, ZCOVER performs fuzzing on the target controller. Here, ZCOVER utilizes *position-sensitive mutation* for effective fuzzing.

The hierarchical structure of the Z-Wave application layer can be visualized as a tree data structure (see Figure 6). The top-level field at position 0 represents the CMDCL, which defines a group of related commands corresponding to specific functionality (*e.g.*, door lock, switch binary, status). At the next level, at position 1, resides the CMD, which specifies a particular action or query within the command class (*e.g.*, *Set*, *Get*, *Report*). Dependent fields (position 2) and beyond are PARAMs, which provide the detailed data required by the command to perform its function, such as operational settings. This structured approach ensures that Z-Wave messages are organized in a clear, modular fashion, facilitating precise communication and control between controller and slave devices.

Mutating the application layer of Z-Wave messages should carefully consider the position of fields to maintain the structural integrity and validity of the communication protocol; incorrectly positioned mutations can lead to invalid frames that devices may reject or ignore, thereby reducing the effectiveness of the fuzzing process.

To address this issue and achieve effective fuzzing, ZCOVER utilizes position-sensitive mutation.

1) Mutation strategies: The key idea of position-sensitive mutation is to effectively mutate the test packet by considering the position of each packet field. ZCOVER previously identified the known and unknown CMDCLs of the controller. A Z-Wave packet has corresponding CMDs and PARAMs according to CMDCLs. ZCOVER focuses on such a correlation, creating packets that have a low probability of being rejected by the target controller and are effective in finding vulnerabilities. Specific strategies are as follows.

TABLE I: Mutation operators assigned to Z-Wave application layer fields.

Field	Len	Mutation operators*
H-ID	4	None
SRC	1	None
P1	1	None
P2	1	None
LEN	1	None
DST	1	None
CMDCL	1	*rand_valid*
CMD	1	*rand_valid, rand_invalid, arith, interesting, insert*
PARAM_1	1	*rand_valid, rand_invalid, arith, interesting, insert*
PARAM_2	1	*rand_valid, rand_invalid, arith, interesting, insert*
...		
PARAM_m	1	*rand_valid, rand_invalid, arith, interesting, insert*
CS	1	None

**rand_valid*: Replace with a randomly selected legal value.
**rand_invalid*: Replace with a randomly selected illegal value.
**arith*: Add/subtract small integer.
**interesting*: Replace with interesting values.
**insert*: Append a random byte.

Mutation field selection. For more efficient mutations, ZCOVER focuses on the *application layer* (*e.g.*, CMDCL, CMD, and PARAMs) of Z-Wave. Note that the maximum size of the Z-Wave MAC layer is 64 bytes (see Section II-A). Thus, randomly mutating all bits in a 64-byte Z-Wave MAC packet would involve the permutation of 512 bits (*i.e.*, 2^{512}), which is an extremely large number to achieve, taking over 4.25×10^{146} years while sending one packet per second.

Additionally, mutations on all Z-Wave packet fields are less effective for the purpose of vulnerability detection. For example, even if we change the values of home ID, checksum, and length, this only impairs the validity of the packet and does not help in identifying hidden vulnerabilities.

ZCOVER addresses this by reducing the input space to the application layer and prioritizing controller-implemented CMDCLs, significantly enhancing efficiency over random MAC layer mutation. Moreover, ZCOVER focuses on the application layer to ensure comprehensive testing of the controller's functionalities, reducing the chances of missing potential vulnerabilities in overlooked packet fields. This can increase the function (*i.e.*, CMDCL) coverage and be efficient in finding new bugs (see Section IV). The previously discovered known and unknown CMDCLs of the Z-Wave controller and their priorities are considered here.

Dynamic and semantic mutation. Instead of applying random mutations blindly, ZCOVER applies mutations that are contextually relevant and meaningful for each parameter: considering the type and semantic meaning of CMDCLs, the corresponding CMDs and PARAMs are dynamically mutated

Algorithm 1: ZCOVER Position-Sensitive Mutator

Input : **C_List**: controller's supported CMDCLs (*e.g.*, 45)
C_T: CMDCL mutation time
Testing_T: fuzzing duration (*e.g.*, 0.1 to 24 hours)
Controller: target Z-Wave controller
Output : **Bug_Logs**: List of bug-inducing packets

```
Initialize Queue with C_List (e.g., test 45 CMDCLs out of 256 )
Set Start_Time ← Current Time
Initialize Bug_Logs as an empty list
while Current Time - Start_Time < Testing_T and Queue is not empty do
    Dequeue the next CMDCL from Queue (e.g., 0x01)
    Set CMD ← 0x00, PARAM ← 0x00
    for each mutation within C_T do
        Generate a semi-valid Packet using CMDCL, CMD, and PARAM (e.g., Initial pld: [0x01 0x00 0x00])
        Send Packet to Controller and observe response
        if Controller hangs or crashes then
            Log Packet into Bug_Logs
        else
            Mutate CMD and PARAM (e.g., 0x0D 0xAA)
    if no crash occurs for the current CMDCL after C_T then
        Move to the next CMDCL in the queue (e.g., 0x7A)
Save Bug_Logs to file for future analysis
return Bug_Logs
```

to create a semi-valid packet that the target controller does not reject. For example, if a parameter represents a controller configuration setting, ZCOVER mutates it within a reasonable range of values to test how the controller responds.

Boundary testing. ZCOVER focuses on mutating parameters near their boundaries and edge cases to uncover potential off-by-one errors, boundary checks, and overflow/underflow conditions. This includes testing minimum and maximum values, as well as values close to these limits.

2) Packet mutation and fuzzing: A Z-Wave controller implements only certain application payload CMDCLs according to its type and properties as revealed during the device known properties fingerprinting (see Section III-B) and unknown properties discovery phases (see Section III-C).

The position-sensitive mutator (PSM) algorithm is a fuzzing method designed to identify vulnerabilities in Z-Wave controllers. It prioritizes both listed and unlisted CMDCLs, generating semi-valid packets by modifying dependent commands and parameters. Each packet is sent to the controller, with crashes or bugs logged for further analysis. If no vulnerabilities are found within a specific time for a CMDCL, PSM moves to the next CMDCL in the queue. This approach ensures efficient testing of a controller's critical functionalities while minimizing wasted effort on unsupported CMDCLs. Algorithm 1 shows the overall PSM process and Table I lists the fields of a Z-Wave packet that ZCOVER mutates coupled with their mutation operators.

Here, dynamic and semantic mutation involve modifying Z-Wave packet values to align with protocol specifications while deviating from typical inputs, exposing potential vulnerabilities. For example, a BINARY Z-Wave packet [0x20, 0x01, 0xFF] (where 0x20 is CMDCL, 0x01 is SET CMD, and 0xFF turns the light on) can be mutated to [0x20, 0x06, 0xFF] (an unsupported CMD) or [0x20, 0x01, 0x00] (a minimum intensity value).

These mutations, combined with boundary testing of extreme values, ensure the controller handles the full range of inputs without failing. Together, these techniques identify weaknesses, such as buffer overflow or underflow, and assess the controller's resilience to sophisticated attacks.

Finally, ZCOVER sends mutated packets to the target controller to identify potential bugs and vulnerabilities. ZCOVER examines the controller response to identify crashes, unexpected responses, or any deviation from the controller's normal behavior. The information collected from this analysis is then fed back to refine the fuzzing process, generating more targeted test cases. This creates a feedback loop that continuously improves the effectiveness of fuzzing. Figure 7 illustrates the high-level workflow of fuzzing.

TABLE II: Tested device details information.

IDX	Brand name	Device type	Model (year)	Encryption support*
D1	ZooZ	Controller	ZST10 (2022)	Yes
D2	SiLab	Controller	UZB-7(2019)	Yes
D3	Nortek	Controller	HUSBZB-1 (2015)	Yes
D4	Aeotec	Controller	ZW090-A (2015)	Yes
D5	ZWaveMe	Controller	ZMEUUZB1 (2015)	Yes
D6	Samsung	Controller	ET-WV520 (2017)	Yes
D7	Samsung	Controller	STH-ETH-200 (2015)	Yes
D8	Schlage †	Door Lock	BE469ZP (2019)	Yes
D9	GE Jasco †	Smart Switch	ZW4201 (2016)	No

*Encryption support: whether or not the device supports data encryption. † Slave devices D8 & D9 are added to create a realistic smart home.

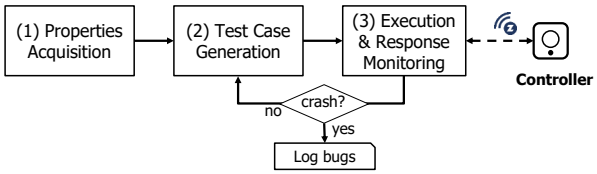


Fig. 7: High-level workflow of the fuzzing process.

IV. EVALUATION

In this section, we evaluate ZCOVER. Section IV-A investigates zero-day vulnerabilities discovered by ZCOVER. Section IV-B presents the efficacy of ZCOVER in assessing Z-Wave controllers. Section IV-C compares ZCOVER with an existing approach [9] to demonstrate its effectiveness. Finally, Section IV-D presents the ablation study.

Implementation of ZCOVER. ZCOVER comprises five modules: a passive scanner and active scanner for retrieving target device properties; a CMDCL cluster for discovering unknown properties; a packet mutator for generating semi-valid test cases; and a packet tester for validating selected packets saved in the log file. All modules are implemented in 980 lines of Python code (excluding external libraries) [21].

Testbed. The system under test (SUT) or testbed consists of real-world Z-Wave devices used in smart homes, including the

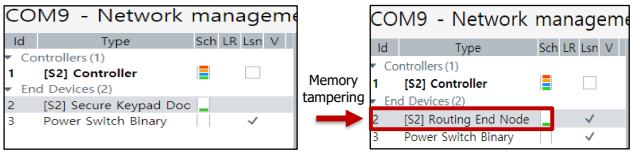


Fig. 8: Controller's memory tampering attack. Smart door lock ID #2 property is changed to routing slave.

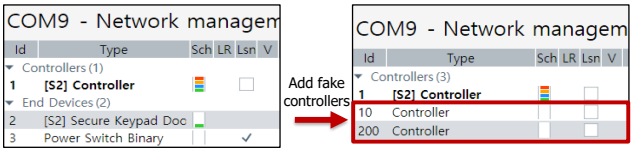


Fig. 9: Controller's memory tampering attack. We inserted rogue (fake) controllers of ID #10 and #200.

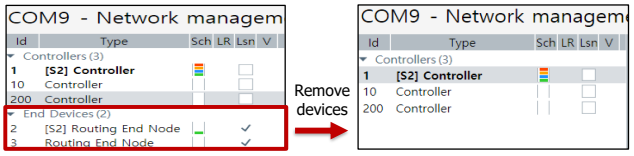


Fig. 10: Controller's memory tampering attack. Removing device ID #2 and ID #3 in the controller's memory.

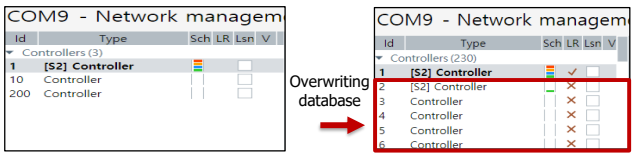


Fig. 11: Overwriting the controller's device table database with fake devices.

latest S2, S0, and legacy devices from various manufacturers. The aim is to achieve a diversified and unbiased evaluation. Table II summarizes detailed information about the devices. To replicate the smart home environment, we included two slave devices `D8` and `D9` in the testbed.

Experiment environment. We ran ZCOVER on a machine with Ubuntu 18.04.6 LTS, an Intel Core i5-7th Gen CPU (2.5 GHz), 8GB of RAM, and a 500GB SSD. The Z-Wave primary controller, *i.e.*, the Samsung SmartThings hub (see Table II), was connected to the Internet and used the SmartThings app installed on an iPhone 15 to remotely control the device via the 5G communication network. Moreover, we used the Yardstick [22] dongle as the Z-Wave transceiver due to its support from the open-source community. The Ubuntu machine running ZCOVER was placed at an average distance of 10 to 70 meters from the target device.

Additionally, we used Z-Wave PC Controller program to control the USB interface controllers (*i.e.*, D1-D5 in Table II). The program ran on a Samsung Galaxy Book (Model NT951XDB) laptop running Microsoft Windows 10, equipped with an Intel Core i7-1165G7 CPU (2.8 GHz), 16GB of RAM, and a 500GB SSD. Following recommended fuzzing practices, we conducted five 24-hour fuzzing trials for each controller. Note that the fuzzing duration can be adjusted manually by the operator.

A. Zero-day vulnerabilities discovery

Table III summarizes the zero-day vulnerabilities discovered by ZCOVER. As a result of applying ZCOVER to the selected target controllers, it successfully discovered 15 zero-day vulnerabilities, 12 of which have been assigned new CVE IDs. The remaining vulnerabilities have also been reported and confirmed.

These vulnerabilities reside in the Z-Wave specification and implementation. Their root causes include a lack of authentication, weak identity verification, inadequate access control, missing packet validation, and unencrypted sensitive data (*e.g.*, `CMDCL 0x01`).

We present case studies regarding key proof-of-concept attack scenarios related to selected critical zero-day vulnerabilities found by ZCOVER. In addition, we provide sample videos in [10], [11] highlighting the impact of vulnerabilities found on real devices.

Bug ID #01: Memory corruption in existing slave device properties. This attack involves manipulating the properties of a slave device stored in the controller's memory. In Figure 8, the initial memory list of our controller includes device ID #2, representing the smart door lock (D8 in Table II) employing the latest S2 security encryption. Despite the utilization of this advanced encryption, we identified a vulnerability exploited by an *unlisted* proprietary `CMDCL 0x01`, responsible for managing the node in the controller memory. This `CMDCL` should undergo encryption for processing; however, we discovered its acceptance of non-encrypted Z-Wave packets, leading us to further investigation. This vulnerability stems from a flaw in the Z-Wave specification. Upon injecting malicious payloads, the properties associated with the smart lock vanish from the controller's memory, resulting in the user being unable to control the door.

Bug ID #02: Fake or rogue device insertion into controller's memory. By exploiting this attack, we can insert fake devices behaving like controllers into the main Z-Wave valid controller (see Figure 9). These rogue devices can compromise security by serving as entry points for attackers, intercepting and manipulating data, and causing system instability or malfunction. They are difficult to detect due to their resemblance to genuine devices.

Bug ID #03: Removing valid device in the controller's memory. Removing a valid device from the smart home controller memory can significantly impact functionality, security, and user experience. For instance, removing the initial device ID #2 (see Figure 10) could disable door automation, create security vulnerabilities, and disrupt automation sequences. This necessitates re-configuring settings and routines, leading to user inconvenience and frustration.

TABLE III: Zero-day vulnerability discovery results of ZCOVER. For ethical reasons, full packet payloads are not disclosed.

Bug ID	Affected devices	CMDCL	CMD	Description	Duration	Root cause	Confirmed
01	D1 - D7	0x01	0x0D	Memory corruption in existing device properties.	Infinite*	Specification	CVE-2024-50929
02	D1 - D7	0x01	0x0D	Fake device insertion into controller's memory.	Infinite	Specification	CVE-2024-50920
03	D1 - D7	0x01	0x0D	Remove valid device in the controller's memory.	Infinite	Specification	CVE-2024-50931
04	D1 - D7	0x01	0x0D	Overwriting the controller's device database.	Infinite	Specification	CVE-2024-50930
05	D6 and D7	0x01	0x02	DoS on smartphone app.	Infinite	Specification	CVE-2024-50921
06	D1 - D5	0x9F	0x01	Z-Wave PC controller program crash.	Infinite	Implementation	CVE-2023-6640
07	D1 - D7	0x5A	0x01	Service interruption during the attack.	68 sec	Specification	CVE-2023-6533
08	D1 - D7	0x59	0x03	Service interruption during the attack.	67 sec	Specification	CVE-2024-50924
09	D1 - D7	0x7A	0x01	Service interruption during the attack.	63 sec	Specification	CVE-2023-6642
10	D1 - D7	0x86	0x13	Service interruption during the attack.	4 sec	Specification	CVE-2023-6641
11	D1 - D7	0x59	0x05	Service interruption during the attack.	62 sec	Specification	CVE-2023-6643
12	D1 - D7	0x01	0x0D	Remove the device's wakeup interval value.	Infinite	Specification	CVE-2024-50928
13	D1 - D5	0x73	0x04	Dos on the Z-Wave PC controller program.	Infinite	Implementation	✓
14	D1 - D7	0x01	0x04	Z-Wave controller service disruption.	4 min	Specification	✓
15	D1 - D7	0x7A	0x03	Service interruption during the attack.	59 sec	Specification	✓

*Infinite: Users cannot control their devices. ✓: Vendors acknowledged the reported bugs.

Bug ID #04: Overwriting the controller's database. Overwriting the smart home controller's device database can severely impact functionality, security, and stability (see Figure 11). This results in the loss of all device configurations, requiring manual reconfiguration. It disrupts automation routines, causing device behavior inconsistencies and potential service interruptions. Without proper backups, critical data can be permanently lost or corrupted, making restoration a complex and time-consuming process.

Bug ID #05: DoS on smartphone app. This attack disrupts system functionality by preventing legitimate users from accessing the smart home. A DoS occurred on the SmartThings app when ZCOVER sent a mutated packet with `CMDCL 0x01` to the controller. During the attack, the homeowner was unable to control the smart switch due to the controller processing the malicious packet.

Bug ID #06: Z-Wave PC controller crash. The Z-Wave PC controller program, used on Windows laptops with USB stick controllers (D1-D5 in Table II), experiences a DoS attack causing it to crash repeatedly. During the attack, users lose control of all devices, and the program only functions normally if the attack stops.

Bug ID #07 to #11 and #15: Service interruption. These attacks exploit vulnerable `CMDCLs` by injecting fuzzed packets into the target controller. Although these `CMDCLs` should require encryption, we discovered that the controller incorrectly processes non-encrypted packets. As a result, injecting malicious payloads rendered the controller unresponsive, preventing legitimate control during the attack.

Bug ID #12: Removal of wake-up interval. This attack exploits vulnerable `CMDCL`, allowing an attacker to remove the wake-up interval value of a device on the target controller. Since the Z-Wave specification does not enforce strict validation, the controller incorrectly processes the malicious request. As a result, the network becomes unresponsive, requiring manual intervention to restore functionality.

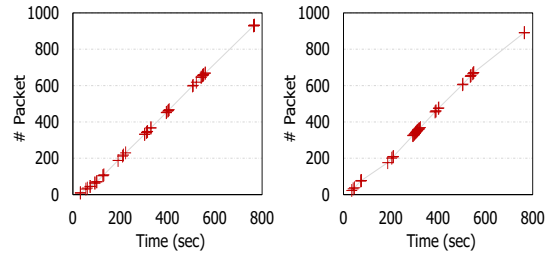
Bug ID #13: DoS on Z-Wave PC Controller. This vulnerability affects Z-Wave PC controller software by exploiting `CMDCL 0x73` and `CMD 0x04`, leading to a persistent DoS. Since the issue persists indefinitely, users lose control of all connected Z-Wave devices, disrupting the smart home system until the software is manually restarted or patched.

Bug ID #14: Z-Wave controller service disruption. We observed a DoS attack on the Z-Wave controller when ZCOVER sent a mutated `WAKEUP CMDCL` packet, causing network disruption for over four minutes. During this time, users lost device control and did not receive intrusion notifications. The attack involved a single packet that kept the controller busy searching for non-existent Z-Wave devices.

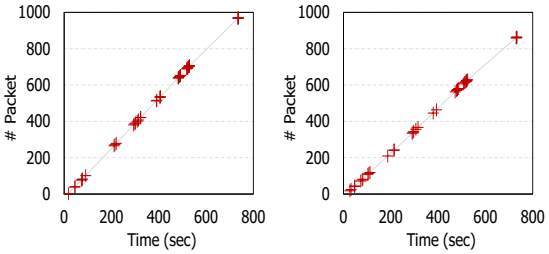
Responsible disclosure. Since November 2023, we have submitted vulnerability reports to the US-CERT/CC division [23] to collaborate with the chipset and device manufacturers responsible for addressing and reducing the risks we identified. All vulnerabilities we reported were confirmed, and 12 vulnerabilities were assigned new CVE IDs.

CERT/CC added more than 16 vendors during the process and Silicon Labs has provided two security advisory reports on the found vulnerabilities along with their remediation. These significant security advisories can be accessed and downloaded after creating a free account and logging into the Silicon Labs platform [24], [25]. Considering ethical concerns, we refrain from disclosing the details of ZCOVER PoC exploit code and payload to prevent potential misuse by malicious actors targeting smart home devices.

Feedback & crash verification. During fuzzing, we assess test cases by monitoring controller liveliness using `NOP` ping packets. Any delays, crashes, or unresponsiveness indicate potential vulnerabilities, which are manually verified due to the closed-source nature of Z-Wave devices. After validation, we develop proof-of-concept (PoC) exploits for selected critical vulnerabilities.



(a) ZooZ (D1) (b) Nortek (D3)



(c) Aeotec (D4) (d) ZWaveMe (D5)

Fig. 12: ZCOVER's vulnerability detection on four devices is shown, with time on the x-axis, test packets on the y-axis, and red crosses marking discoveries. The experiment lasts 24 hours, but the figures highlight the initial fuzzing phase where most of the 15 *unique* zero-day vulnerabilities, including duplicates, were detected.

B. *Efficacy of* ZCOVER

1) Fingerprinting and property discovery: Table IV summarizes ZCOVER's fingerprinting and its discovery of unknown properties. ZCOVER identified nearly twice as many unknown CMDCLs as known ones and effectively fingerprinted target controllers by extracting key properties (*e.g.*, home ID, node ID, CMDCLs) for subsequent security analysis. This result also demonstrated that our property extraction techniques are device-independent and unrestricted.

2) Performance: We evaluated ZCOVER's efficiency during fuzzing by examining the number of vulnerabilities discovered over time and the number of test cases. Figure 12 presents the experimental results.

The first thing to note is that many vulnerabilities were discovered in the early stages of fuzzing (*e.g.*, less than 600 seconds) owing to the approach of prioritizing CMDCLs used by ZCOVER. Additionally, ZCOVER was able to detect numerous vulnerabilities within a relatively short period of time. Unlike traditional fuzzing, which typically tests target devices using millions of packets over several hours, ZCOVER successfully identified many vulnerabilities within an average of 600 seconds and 800 test packets by considering both known and unknown CMDCLs and performing position-sensitive mutation. This result suggests that ZCOVER can effectively discover vulnerabilities in the controller of the real-world Z-Wave smart home systems within a short period of time.

TABLE IV: Controllers' known properties fingerprinting and unknown properties discovery results by ZCOVER.

ID	Passive scanning		Active scanning	Unknown CMDCLs
	Home ID	Node ID	Known CMDCLs	
D1	E7DE3F3D	0x01	17 CMDCLs	28 CMDCLs
D2	CD007171	0x01	17 CMDCLs	28 CMDCLs
D3	CB51722D	0x01	15 CMDCLs	30 CMDCLs
D4	C7E9DD54	0x01	17 CMDCLs	28 CMDCLs
D5	F4C3754D	0x01	15 CMDCLs	30 CMDCLs
D6	CB95A34A	0x01	17 CMDCLs	28 CMDCLs
D7	EDC87EE4	0x01	15 CMDCLs	30 CMDCLs

TABLE V: CMDCL coverage and *unique* vulnerability discovery results (#Vul.) on **controllers** from VFUZZ and ZCOVER. The fuzzing was performed for 24 hours.

ID	VFUZZ			ZCOVER		
	CMDCL*	CMD	#Vul.	CMDCL†	CMD	#Vul.
D1	256	256	1	45	53	**15**
D2	256	256	3	45	53	**15**
D3	256	256	0	45	53	**15**
D4	256	256	4	45	53	**15**
D5	256	256	0	45	53	**15**

*Whole range of 256 CMDCLs covered by VFUZZ.
†**45 CMDCLs** (known and unknown) are prioritized by ZCOVER.

C. *Comparison with an existing approach*

Here, we compared ZCOVER with an existing technique for enhancing Z-Wave smart home security. Although various related approaches exist, there were instances where (1) we could not use the closed-source tools despite multiple requests to the authors, and (2) the approaches could not be applied to the Z-Wave controllers. Therefore, we performed a direct comparison between ZCOVER and VFUZZ [9], which is a recent security research on the Z-Wave protocol.

In the experiment, we applied VFUZZ to devices D1 through D5 (see Table II) in the same environment used in our ZCOVER experiments. We performed fuzzing for 24 hour with both ZCOVER and VFUZZ, and examined the detected *unique* vulnerabilities.

Table V summarizes the experimental results. First of all, we observed that ZCOVER can detect significantly more unique vulnerabilities compared to VFUZZ. Specifically, all vulnerabilities detected by ZCOVER were previously unknown, whereas the results of VFUZZ included known one-day vulnerabilities. Interestingly, according to our manual analysis, there were no vulnerabilities found in common between both tools. This is the result of differences in the Z-Wave packet fields targeted for the mutation (*e.g.*, VFUZZ focuses on the MAC frame of the Z-Wave packets) and differences in mutation methods.

The primary reason for the difference in the number of detected vulnerabilities lies in whether the properties of the controller were adequately considered. As indicated in Table V, VFUZZ generated test packets using a broader range of

CMDCLs and CMDs. Hence, many of the test packets generated by VFuzz failed to assess efficiently the application layer implementation of the target controller, resulting in less bugs, thereby reducing the efficiency of fuzzing.

On the other hand, ZCover identifies both known and unknown proprietary CMDCLs of the target controller (*i.e.*, **45 CMDCLs**) and performs mutations considering the correlation between CMDCLs and CMDs. This enabled the generation of more effective test packets within the same timeframe, leading to the detection of more vulnerabilities.

By focusing on the application layer and effectively identifying controller properties, while using mutations that consider correlations between packet fields, ZCover outperforms existing Z-Wave security techniques in detecting vulnerabilities, demonstrating its effectiveness.

D. Ablation study

To assess the impact of ZCover core techniques, we conducted an ablation study by enabling and disabling key features during fuzzing. The study aimed to measure the effectiveness of known and unknown CMDCLs discovery and position-sensitive mutation in vulnerability detection. By systematically adjusting these parameters, we evaluated how each contributed to ZCover's overall performance.

In our experiment, we ran ZCover for one hour under three different configurations:

- **ZCover full functionality.** Enabled both **known** and **unknown** CMDCLs discovery, and position-sensitive mutation of CMDCL's correlated CMD and PARAMs.
- **ZCover β fuzzing with only known CMDCLs.** Used position-sensitive mutation while ignoring **unknown** CMDCLs.
- **ZCover γ fuzzing with random mutation.** Selected CMDCLs, CMD, and PARAM values randomly without considering ZCover core features.

The results in Table VI highlight the advantage of ZCover targeted and structured mutation approach, confirming that its methodology significantly improves fuzzing efficiency and vulnerability detection. The findings confirm that full ZCover functionality maximizes vulnerability detection, demonstrating the value of discovering hidden CMDCLs and leveraging structured, position-sensitive mutation. Fuzzing with only known CMDCLs identified vulnerabilities but missed critical flaws in unlisted properties, while random fuzzing was the least effective, reinforcing the need for ZCover systematic approach. These results validate the rationale behind ZCover design choices and its practical effectiveness in security analysis on Z-Wave controller.

TABLE VI: Results of the ablation study on ZCover core features and *unique* vulnerability discovery found (#Vul.). The fuzzing was performed for one hour on Zooz controller.

Test	Fuzzing Configuration	#Vul.
1	ZCover full (Known + Unknown CMDCLs + Position-Sensitive Mutation)	**15**
2	ZCover β (Known CMDCLs Only + Position-Sensitive Mutation)	8
3	ZCover γ (Random CMDCLs + No Position-Sensitive Mutation)	6

V. Discussion

A. Research importance

With over 100 million devices globally [9], the Z-Wave ecosystem faces security challenges, as legacy devices lack protection and S2 devices provide only partial security. ZCover addresses these issues by assessing the central controller's security, enhancing the overall Z-Wave network. By uncovering 15 unknown vulnerabilities, ZCover proves its practicality, enabling safer device development and a more secure smart home environment for users.

B. Attack remediation

To address discovered vulnerabilities, S2 devices should block malicious payloads via updated Z-Wave specifications. For legacy devices, a lightweight intrusion detection system (IDS) (*e.g.*, [15]) can detect attacks and trigger alarms or alerts. Easy firmware updates should be supported to patch future vulnerabilities, and manufacturers must prioritize security during device design.

We collaborated with Silicon Labs (SiLabs) and vendors to address these issues. SiLabs confirmed mitigation plans for recent devices and announced a Z-Wave SDK update to secure additional devices. They also issued two security advisories, and the Z-Wave Alliance will incorporate our findings in the next Z-Wave specification update.

C. Threats to validity

The validity of ZCover's findings may be influenced by several factors. First, the proprietary nature of Z-Wave controllers limits our understanding of the information available through public documentation and reverse engineering. Therefore, hidden or undocumented behaviors could affect the comprehensiveness of our vulnerability assessment. Second, the fuzzing approach of ZCover might not cover all possible edge cases, especially those that require specific environmental conditions or sequences of operations to manifest. Third, the hardware and software configurations of Z-Wave controllers can vary across manufacturers and models. However, because all Z-Wave devices from different manufacturers must have a Z-Wave chipset onboard for interoperability, our findings apply to all manufacturers.

D. Limitations

Although ZCover has discovered a number of unknown vulnerabilities in real Z-Wave controllers, there are several limitations that could be improved.

First, due to the closed-source nature of Z-Wave controller firmware and proprietary documentation, ZCover might miss certain vulnerabilities or misinterpret some device behaviors

that could be better understood with full access to the source code and detailed technical specifications. Second, as ZCOVER operates as an external entity without privileged access to the network, it relies on external sniffing and packet injection techniques. This approach may not capture all nuances of the internal processing of the controller, potentially overlooking some subtle security issues. Last, while ZCOVER automates much of the testing process, interpreting the implications of discovered vulnerabilities may require significant manual effort and expert knowledge. These limitations highlight areas where ZCOVER can be improved and underscore the importance of using it as part of a broader security assessment strategy that includes multiple tools and approaches. Despite these limitations, we believe that ZCOVER provides a robust framework for enhancing the security of Z-Wave controllers.

VI. Related works

Several studies have been conducted to identify Z-Wave vulnerabilities (*e.g.*, [6], [7], [17], [26]–[28]). For example, Fouladi and Ghanoun [7] examined the security of the Z-Wave protocol by focusing on its encryption mechanisms. However, these studies are not applicable to discover the vulnerabilities within the controller's application layer. Similarly, several studies have been conducted to increase Z-Wave security (*e.g.*, [29]–[35]), but these have focused on attacks that can occur in communication rather than application layer vulnerabilities, thus they cannot be applied to detecting unknown vulnerabilities in the Z-Wave controller.

There are two studies that focus on the internal vulnerabilities of Z-Wave devices (*e.g.*, [5], [9]). Xiaoyue *et al.* [5] presents HubFuzzer, a fuzzing technique for discovering vulnerabilities in IoT devices that communicate using ZigBee and Z-Wave protocols. However, HubFuzzer runs on a Z-Wave controller to test slave devices; in other words, it cannot be applied to detect vulnerabilities in the Z-Wave controller. Nkuba *et al.* [9] introduced VFUZZ, a fuzzing framework for testing Z-Wave devices. While this work is effective for general Z-Wave analysis, its broader approach does not provide the depth and specificity required to thoroughly assess the Z-Wave controller's application layer implementation (see Section IV-C).

Other research investigated the security of IoT devices and other protocols (*e.g.*, IP, Wi-Fi, Bluetooth, and ZigBee [36]–[43]). While the techniques could be adapted for Z-Wave, the research did not specifically target Z-Wave, thereby failing to address application layer vulnerabilities and unknown command classes specific to Z-Wave.

ZCOVER highlights the practical impact of these vulnerabilities through proof-of-concept attack scenarios on a real Z-Wave smart home setup, emphasizing the need for better security testing and hardening of Z-Wave devices.

VII. Conclusion

As the Z-Wave protocol is widely used as a key feature of smart home systems, issues regarding its security are increasing. In this paper, we discover the vulnerabilities contained in the Z-Wave smart home controllers by leveraging our new security testing framework called ZCOVER. By identifying hidden properties (*e.g.*, unknown `CMDCL`s) as well as known properties (*e.g.*, home ID and listed `CMDCL`s) of the target Z-Wave controller and utilizing position-sensitive mutation that considers the correlation of Z-Wave packet frame fields, ZCOVER improves the security and robustness of Z-Wave target devices. Our evaluation results revealed that ZCOVER could discover critical vulnerabilities in Z-Wave controllers, by detecting **15 new zero-day vulnerabilities**. Through ZCOVER, Z-Wave device developers can improve the devices' security, and Z-Wave smart home users can enjoy more secure services. ZCOVER is publicly available at [21] and will be serviced on https://iotcube.net.

VIII. Acknowledgments

We are grateful to all anonymous reviewers for their valuable feedback that helped improve this paper. This work was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP), grant funded by the Korea government (MSIT) (No. RS-2022-II220277 Development of SBOM Technologies for Securing Software Supply Chains, No. RS-2024-00440780 Development of Automated SBOM and VEX Verification Technologies for Securing Software Supply Chains, and No. IITP-2025-RS-2020-II201819 ICT Creative Consilience Program. In addition, this work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (RS-2025-00517788, Research on Intelligent SBOM Generation and Automated Vulnerability Analysis through Multi-level Code Analysis). Finally, this research was supported by Culture, Sports and Tourism R&D Program through the Korea Creative Content Agency grant funded by the Ministry of Culture, Sports and Tourism (Project Name: International Collaborative Research and Global Talent Development for the Development of Copyright Management and Protection Technologies for Generative AI, Project Number: RS-2024-00345025).

References

[1] Z-Wave Alliance. (2023) Smart home control on one app. [Online]. Available: https://www.z-wave.com/

[2] K. Kafle, K. Jagtap, M. Ahmed-Rengers, T. Jaeger, and A. Nadkarni, "Practical Integrity Validation in the Smart Home with HomeEndorser," in *Proceedings of the 17th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, 2024, pp. 207–218.

[3] C. Fu, X. Du, Q. Zeng, Z. Zhao, F. Zuo, and J. Di, "Seeing is Believing: Extracting Semantic Information from Video for Verifying IoT Events," in *Proceedings of the 17th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, 2024, pp. 101–112.

[4] S. Sunar, P. Shirani, S. Majumdar, and J. D. Brown, "On Continuously Verifying Device-level Functional Integrity by Monitoring Correlated Smart Home Devices," in *Proceedings of the 17th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, 2024, pp. 219–230.

[5] X. Ma, Q. Zeng, H. Chi, and L. Luo, "No More Companion Apps Hacking but One Dongle: Hub-Based Blackbox Fuzzing of IoT Firmware," in *Proceedings of the 21st Annual International Conference on Mobile Systems, Applications and Services*, 2023, pp. 205–218.

[6] C. Badenhop, S. R. Graham, B. W. Ramsey, B. E. Mullins, and L. O. Mailloux, "The Z-Wave routing protocol and its security implications," *Computers & Security*, vol. 68, pp. 112–129, 2017.

[7] B. Fouladi and S. Ghanoun, "Security evaluation of the Z-Wave wireless protocol," *BlackHat USA*, vol. 24, pp. 1–2, 2013.

[8] J. L. Hall, "A Practical Wireless Exploitation Framework for Z-Wave Networks," Air Force Institute of Technology Wright-Patterson AFB OH Wright-Patterson, Tech. Rep., 2016.

[9] C. K. Nkuba, S. Kim, S. Dietrich, and H. Lee, "Riding the IoT Wave With VFuzz: Discovering Security Flaws in Smart Homes," *IEEE Access*, vol. 10, pp. 1775–1789, 2022.

[10] C. K. Nkuba and K. Jimin. ZCover Proof-of-Concept Video 1. [Online]. Available: https://drive.google.com/file/d/1LBycOFbQThFxuGedefVfNqNa0TbTE0R0/view

[11] C. K. Nkuba and K. Jimin. ZCover Proof-of-Concept Video 2. [Online]. Available: https://drive.google.com/file/d/1aZMcGRUVtweYkWlcHzWRsl1jhp1nSBYs/view

[12] Silicon Laboratories. (2023) Z-Wave Specification. [Online]. Available: https://www.silabs.com/products/wireless/mesh-networking/z-wave/specification

[13] Z-Wave Alliance. (2023) 2023 Ecosystem Report. [Online]. Available: https://z-wavealliance.org/zwave-ecosystem-report/

[14] C. Paetz, *Z-Wave Essentials*. CreateSpace Independent Publishing Platform, 2018.

[15] C. K. Nkuba, S. Woo, H. Lee, and S. Dietrich, "ZMAD: Lightweight Model-Based Anomaly Detection for the Structured Z-Wave Protocol," *IEEE Access*, vol. 11, pp. 60 562–60 577, 2023.

[16] CERT Coordination Center. (2022) Silicon Labs Z-Wave chipsets contain multiple vulnerabilities. [Online]. Available: https://kb.cert.org/vuls/id/142629

[17] C. Badenhop, J. Fuller, J. Hall, B. Ramsey, and M. Rice, "Evaluating ITU-T G. 9959 Based Wireless Systems used in Critical Infrastructure Assets," in *International Conference on Critical Infrastructure Protection*. Springer, 2015, pp. 209–227.

[18] B. P. Miller, L. Fredriksen, and B. So, "An empirical study of the reliability of UNIX utilities," *Communications of the ACM*, vol. 33, no. 12, pp. 32–44, 1990.

[19] Z-Wave Alliance. Application Work Group Z-Wave Specification Release 2023B. [Online]. Available: https://sdomembers.z-wavealliance.org/document/dl/917

[20] Milind Dumbare. Z-Wave public. [Online]. Available: https://github.com/Z-WavePublic/libzwaveip/blob/master/config/ZWave_custom_cmd_classes.xml

[21] C. K. Nkuba and K. Jimin. ZCover Public Version. [Online]. Available: https://github.com/CNK2100/ZCOVER_PUBLIC

[22] G. S. Gadgets, "YARD Stick One-a sub-1 GHz wireless test tool controlled by your computer," 2021. [Online]. Available: https://greatscottgadgets.com/yardstickone/

[23] Software Engineering Institute. The CERT Division. https://www.sei.cmu.edu/about/divisions/cert/index.cfm.

[24] Silicon Labs. Security Advisories A-00000502. [Online]. Available: https://community.silabs.com/068Vm000001HdNm

[25] Silicon Labs. Security Advisories A-00000505. [Online]. Available: https://community.silabs.com/068Vm00000211lw

[26] K. Kim, K. Cho, J. Lim, Y. ho Jung, M. S. Sung, S. B. Kim, and H. K. Kim, "What's your protocol: Vulnerabilities and security threats related to Z-Wave protocol," *Pervasive and Mobile Computing*, vol. 66, no. 101211, 2020.

[27] J. D. Fuller, "A Misuse-Based Intrusion Detection System for ITU-T G. 9959 Wireless Networks," Air Force Institute of Technology Wright-Patterson AFB OH Wright-Patterson, Tech. Rep., 2016.

[28] J. D. Fuller, B. W. Ramsey, M. J. Rice, and J. M. Pecarina, "Misuse-based detection of Z-Wave network attacks," *Computers & Security*, vol. 64, pp. 44–58, 2017.

[29] N. Boucif, F. Golchert, A. Siemer, P. Felke, and F. Gosewehr, "Crushing the Wave: New Z-Wave vulnerabilities exposed," *arXiv preprint arXiv:2001.08497*, 2020.

[30] J.-M. Picod, A. Lebrun, and J.-C. Demay, "Bringing software defined radio to the penetration testing community," in *Black Hat USA Conference*, 2014.

[31] N. Apthorpe, D. Y. Huang, D. Reisman, A. Narayanan, and N. Feamster, "Keeping the Smart Home Private with Smart(er) IoT Traffic Shaping," in *Proceedings of the 2019 Privacy Enhancing Technologies Symposium (PETS)*, 2018, pp. 1—21.

[32] C. Badenhop and B. Ramsey, "Carols of the Z-Wave security layer; Or, robbing keys from Peter to unlock Paul," *PoC or GTFO*, vol. 12, pp. 6–12, 2016.

[33] J. D. Fuller and B. W. Ramsey, "Rogue Z-Wave controllers: A persistent attack channel," in *2015 IEEE 40th Local Computer Networks Conference Workshops (LCN Workshops)*. IEEE, 2015, pp. 734–741. [Online]. Available: https://doi.org/10.1109/LCNW.2015.7365922

[34] L. Babun, H. Aksu, L. Ryan, K. Akkaya, E. S. Bentley, and A. S. Uluagac, "Z-IoT: Passive Device-class Fingerprinting of ZigBee and Z-Wave IoT Devices," in *IEEE International Conference on Communications 2020 (ICC)*. IEEE, 2020, pp. 1–7.

[35] J.-C. Liou, S. Jain, S. R. Singh, D. Taksinwarajan, and S. Seneviratne, "Side-channel information leaks of Z-Wave smart home IoT devices: Demo abstract," in *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*, 2020, pp. 637–638.

[36] H. Gollier and M. Vanhoef, "SSID Confusion: Making Wi-Fi Clients Connect to the Wrong Network," in *Proceedings of the 17th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, 2024, pp. 156–161.

[37] Y. Yu and J. Liu, "TAPInspector: Safety and Liveness Verification of Concurrent Trigger-Action IoT Systems," *arXiv preprint arXiv:2102.01468*, 2021.

[38] Z. B. Celik, G. Tan, and P. D. McDaniel, "IoTGuard: Dynamic Enforcement of Security and Safety Policy in Commodity IoT," in *NDSS Symposium*, 2019, pp. 1–15.

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

[40] Y. Jia, L. Xing, Y. Mao, D. Zhao, X. Wang, S. Zhao, and Y. Zhang, "Burglars' IoT Paradise: Understanding and Mitigating Security Risks of General Messaging Protocols on IoT Clouds," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 465–481.

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

[42] F. Heiding, E. Süren, J. Olegård, and R. Lagerström, "Penetration testing of connected households," *Computers & Security*, vol. 126, p. 103067, 2023.

[43] H. Park, C. K. Nkuba, S. Woo, and H. Lee, "L2Fuzz: Discovering Bluetooth L2CAP Vulnerabilities Using Stateful Fuzz Testing," in *2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2022, pp. 343–354.