Received November 1, 2021, accepted December 11, 2021, date of publication December 27, 2021, date of current version January 6, 2022.

Digital Object Identifier 10.1109/ACCESS.2021.3138768

Riding the IoT Wave With VFuzz: Discovering Security Flaws in Smart Homes

CARLOS KAYEMBE NKUBA[1], **SEULBAE KIM**[2], **SVEN DIETRICH**[3], **(Senior Member, IEEE), AND HEEJO LEE**[1]

[1]Department of Computer Science and Engineering, Korea University, Seoul 02841, Republic of Korea
[2]Department of Computer Science, Georgia Institute of Technology, Atlanta, GA 30332, USA
[3]Department of Computer Science, Hunter College, City University of New York (CUNY), New York, NY 10065, USA

Corresponding author: Heejo Lee (heejo@korea.ac.kr)

This work was supported in part by the Institute of Information & Communications Technology Planning & Evaluation (IITP) Grant funded by the Korea Government [Ministry of Science and Information and Communications Technology (ICT) (MSIT)] through the Development of Automated Vulnerability Discovery Technologies for Blockchain Platform Security under Grant 2019-0-01697, in part by the Regional Strategic Industry Convergence Security Core Talent Training under Grant 2019-0-01343, in part by the ICT Creative Consilience Program under Grant IITP-2021-2020-0-01819, and in part by the Research Foundation City University of New York (RFCUNY).

ABSTRACT Z-Wave smart home Internet of Things devices are used to save energy, increase comfort, and remotely monitor home activities. In the past, security researchers found Z-Wave device vulnerabilities through reverse engineering, manual audits, and penetration testing. However, they did not fully use fuzzing, which is an automated cost-effective testing technique. Thus, in this paper, we present VFuzz, a protocol-aware blackbox fuzzing framework for quickly assessing vulnerabilities in Z-Wave devices. VFuzz assesses the target device capabilities and encryption support to guide seed selection and tests the target for new vulnerability discovery. It uses our field prioritization algorithm (FIPA), which mutates specific Z-Wave frame fields to ensure the validity of the generated test cases. We assessed VFuzz on a real Z-Wave network consisting of 19 Z-Wave devices ranging from legacy to recent ones, as well as different device types. Our VFuzz evaluation found 10 distinct security vulnerabilities and seven crashes among the tested devices and yielded six unique common vulnerabilities and exposures (CVE) identifiers related to the Z-Wave chipset.

INDEX TERMS Smart home security, Z-Wave, Internet of Things, fuzzing, vulnerabilities discovery.

I. INTRODUCTION

The number of Internet of Things (IoT) devices is expected to increase exponentially every year [1]. The IoT smart home automation industry follows this trend: more than 100 million Z-Wave chipset modules have been sold to smart home service providers [2]. This is due to Z-Wave wireless smart home protocols [3] being an appealing choice for several device manufacturers because of their simplicity of use, interoperability among different devices, power efficiency, backward compatibility with legacy devices [4], and use of a frequency range below 1 GHz that does not interfere with other common wireless protocols, *e.g.*, Wi-Fi frequency band of 2.4 GHz.

Despite the rapid growth of Z-Wave smart home devices, manufacturers tend to focus more on device functionality than on security. Consequently several serious flaws have been reported in the past [5]–[17]. Moreover, device vendors did not inform the users about the weaknesses or shortcomings of their *legacy* products, which did not implement encryption; thus, they are vulnerable to remote control, as demonstrated in Section V and in [12]. In fact, we found that only 27% of the Z-Wave products available worldwide [18] implemented and supported the latest Z-Wave Security 2 (S2) encryption mechanism that protects against replay attacks. Despite the security enhancement in S2 devices, attack vectors with critical security implications, *e.g.*, completely neutralizing the S2 controller and alarm system, still exist, as Section V demonstrates. However, information on device vulnerabilities or known common vulnerabilities and exposures (CVEs) for Z-Wave products [19], [20] are scarce, which encouraged us to investigate the Z-Wave ecosystem. We intend to provide not only security awareness to end users about their smart home devices, but also to enable device manufacturers to fix the discovered flaws.

Recent and past studies on Z-Wave security found device flaws by relying on manual techniques, such as reverse

The associate editor coordinating the review of this manuscript and approving it for publication was Zheng Yan.

engineering, manual audits, risk analysis, penetration testing, and chipset memory extraction, which are not only time-consuming, but complex to implement [16], [17], involving significant computation cost [11], and difficulties in reproducing results [8], [9]. These approaches failed to meet the imminent need to promptly analyze the security of commercialized products, as finding and fixing issues early on is key to preventing attackers from abusing and collecting data from smart homes, harassing users, illegaly accessing homes, and hijacking the smart home gateway to launch IoT-botnet-based distributed denial-of-service (DDoS) attacks on vulnerable critical cyberinfrastructures [21]–[25]. Moreover, existing fuzzers such as AFL [26] cannot fuzz IoT devices because of the lack of hardware support, the non-availability of device source code, firmware, and memory debug analysis tools.

To keep up with the fast development pace and facilitate the testing of Z-Wave devices for bugs and security vulnerabilities, we apply a fuzzing approach that is highly effective in finding new vulnerabilities at a low cost to systematically assess Z-Wave. We present **VFuzz** (Z-WaVe protocol **Fuzz**er), which is a feedback-driven fuzzing framework that features a semantic-aware mutation of input packets using the domain knowledge of Z-Wave, a cyber-physical executor that automatically orchestrates remote Z-Wave devices for testing, and a state watchdog consisting of a sensor to monitor device states and generate feedback. With VFuzz, the entire process of testing a Z-Wave device is fully automated: (1) turning the device on and initializing; (2) analyzing device capabilities and encryption supports to guide the seed selection; (3) generating and mutating semi-valid packets; (4) sending the packet to the device; (5) monitoring the states and bugs; and (6) capturing feedback to guide the input mutator.

To evaluate VFuzz, we used 19 different Z-Wave devices from different manufacturers to build a diverse, realistic smart home environment for testing. The evaluation results demonstrate that VFuzz effectively detects 10 distinct device security vulnerabilities and seven crashes among the tested devices with six new CVE identifiers assigned by the US CERT/CC division [27]. Furthermore, the evaluation shows that the tested devices are vulnerable to command injection, data tampering, impersonation, and denial of service (DoS) attacks.

Contributions. This paper makes the following contributions:

- *New semantic-aware mutations.* We propose a practical protocol-aware mutation algorithm called the field prioritization algorithm (FIPA), which makes use of both the syntactic and semantic information of the Z-Wave protocol to generate semi-valid test cases to increase the effectiveness of fuzzing.
- *New IoT fuzzer.* We built the first functional blackbox fuzzer for the Z-Wave protocol that brings the mutation, device orchestration, test execution, and state analysis under one umbrella. Any Z-Wave device can be assessed with low complexity using VFuzz.
- *Zero-day vulnerabilities.* We validated VFuzz on a real Z-Wave test network consisting of both the latest and legacy Z-Wave devices. We found 10 security vulnerabilities that resulted in six new CVEs, assigned by the US CERT/CC, related to the Z-Wave chipset. This study's findings provide awareness to manufacturers to fix and patch vulnerable products. A demonstration video of the impact of found vulnerabilities on smart home devices is available in [28].

The remainder of the paper is organized as follows: Section II presents related work. Section III introduces the Z-Wave protocol information and threat model. Section IV describes our methodology. Section V presents the fuzzer evaluation and results. Section VI describes the discussion and countermeasures and Section VII concludes the paper.

II. RELATED WORK

This section presents past research related to IoT protocol fuzzing and Z-Wave security.

A. IoT FUZZING STUDIES

Fuzzing, in use since the 1990s [29], has been used to discover vulnerabilities especially in operating system (OS) kernels, network protocols, and applications. However, IoT fuzzing has not seen a rapid expansion mainly because of device constraints, such as limited resources and processing power, which result in low fuzzing throughput. Muench *et al.* [30] stated that fuzzing embedded devices is challenging and complex compared to desktop systems because of fault detection, fuzzing performance, and instrumentation challenges owing to the lack of crash-reporting functionalities, multi-processing or virtualization, and firmware source code.

Despite the constraints of fuzzing on embedded devices, several fuzzing test suites have been proposed for IoT systems. KillerBee [31] is a penetration testing tool for the ZigBee protocol. The authors of [32] developed a fuzzer that targets the 6LoWPAN protocol. IoTFuzzer [33] is an app-based fuzzing framework developed as a mobile app that checks for the memory corruption of target IoT devices. Commercial solutions such as BeStorm [34] and Synopsys Defensics [35] do not fuzz the Z-Wave protocol, but instead target ZigBee, CoAP [36], MQTT [37], Bluetooth, and Wi-Fi in their IoT fuzzing test suites.

The state-of-the-art AFL fuzzer [26] cannot fuzz IoT devices owing to the lack of IoT device hardware support. Work by Zheng *et al.* [38] proposed a complex user-mode and full system-mode emulation to fuzz IoT firmware using AFL. This work's limitation was that it was complex to implement and supported only a few CPU architectures and IoT firmware, as it relied on Firmadyne [39] for emulation. Also, Avatar [40] and Muench *et al.* [30] explored process emulation and real hardware to fuzz embedded devices; however, the fuzzing throughput was low.

In summary, existing smart home protocol fuzzers do not assess Z-Wave devices. The specialty of our work is that (1) we conduct fuzzing directly via radio frequencies (RF) on real Z-Wave devices, and (2) we achieve that without the need of a complex emulated system, while increasing fuzzing throughput simultaneously.

B. Z-WAVE SECURITY RESEARCH

In addition to the above fuzzing studies on IoT, some studies have been conducted to find vulnerabilities in Z-Wave smart home devices without fuzzing. In 2013, Z-Force [5] exposed Z-Wave device breaches for the first time using a sniffing device and successfully identified vulnerabilities in a Z-Wave door lock. Scapy-Radio [41], a project from 2014 combining Scapy, GNU Radio Companion (GRC) software, and software-defined radio (SDR), successfully disabled an alarm by injecting the OFF command. Based on Scapy-Radio, the authors in [7], [42], [43] produced a tool called EZ-Wave, which is used for network discovery and device status information gathering.

Prior to our work, other researchers assessed the Z-Wave protocol for device security and privacy assurance. Research in [6]–[17] has revealed protocol implementation vulnerabilities, device non-volatile memory extraction, Z-Wave network key retrieval, rogue controller insertion, identification of Z-Wave threats, and DoS on the Z-Wave controller. These works provide a strong background on Z-Wave vulnerability testing; however, (1) unlike fuzzing, they rely heavily on manual analysis, which is time-consuming, and (2) many listed works do not provide an easy and ready to use system that could help smart home users and device vendors assess their device vulnerabilities.

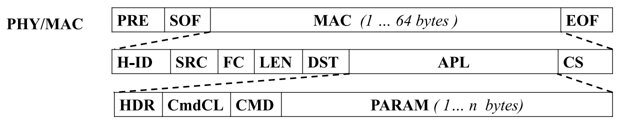
Our research builds on these previous studies and its novelty is the development of an efficient Z-Wave fuzzer for not only fuzz testing, but also for the security testing analysis. Moreover, our fuzzing framework is new in the IoT home automation protocol sphere as it targets the Z-Wave protocol.

III. BACKGROUND AND THREAT MODEL

In this section, we present a brief overview of the Z-Wave protocol and its security features to better understand its functionality. Also, we describe the scope of our research in the context of the larger Z-Wave ecosystem while presenting a threat model related to smart homes.

A. OVERVIEW OF Z-WAVE PROTOCOL

Z-Wave [3] is a wireless home automation protocol developed in 2001 with an alliance of over 800 companies manufacturing over 3300 certified interoperable products worldwide [4]. A single Z-Wave home control network can have up to 232 smart devices interconnected in a mesh topology using the 908 MHz or 916 MHz frequency band for the US, the 868.40 MHz or 869.85 MHz frequency band for Europe, and other frequencies in other parts of the world [44]. The communication range between devices is approximately 30 m indoors and approximately 100 m outdoors.



PRE	Preamble for synchronization	**LEN**	Frame length, 1 byte
SOF	Start of frame, 1 byte	**DST**	Node-ID of the receiver, 1 byte
MAC	MAC data	**APL**	Application payload : Header (HDR), Command Class (CmdCL), Command (CMD), Parameter (PARAM)
H-ID	Home-ID of network, 4 bytes	**CS**	Checksum, 1 byte
SRC	Node-ID of transmitter, 1 byte	**EOF**	End of frame, 1 byte
FC	Frame control flags, 2 bytes		

FIGURE 1. **Z-Wave basic frame structure.**

Device identification and standardization are defined in several classes that specify roles and functionality to ensure interoperability between devices from different vendors in the Z-Wave home control network [45]. Starting in September 2020, the Z-Wave Alliance introduced a new Z-Wave Long Range (LR) specification [46] that increased the wireless range and coverage to 1,609 m, the number of devices per network from 232 to 4000 and is backward compatible and interoperable with legacy-certified Z-Wave devices. Z-Wave LR offers new adoption in industries such as hotels, offices, commercial lights, and multi-dwelling units.

The Z-Wave protocol comprises four layers: physical (PHY), media access control (MAC), network (NWK), and application (APL) layers. The protocol implements the open International Telecommunication Union Standardization (ITU-T) Sector G.9959 at the PHY and MAC layers [47]. The PHY layer manages the frequency selection, modulation, encoding, and decoding of the data. The PHY/MAC layers define the frame structure by adding the preamble (PRE), start-of-frame (SOF), and end-of-frame (EOF) delimiter to allow the receiver to decode the Z-Wave frame. The MAC layer manages collision avoidance, frame acknowledgment (ACK) and frame re-transmission. It contains valuable information for the Z-Wave device communication, such as the home-ID (H-ID), source (SRC), frame control (FC), length (LEN), destination (DST), application layer payload, and checksum (CS) [48].

The NWK layer manages network management and network services to the APL layer *i.e.*, device inclusion and exclusion. The APL layer provides the definition of the frame application payload consisting of the header (HDR), command class (CmdCL), command (CMD), and parameter values (PARAM). It defines the Z-Wave device type and role and provides the transport encapsulation service. Moreover, it offers interoperability and customization between different Z-Wave device manufacturers [49]. The total Z-Wave frame size, including the PRE, SOF, MAC, and EOF, is between 24 and 76 bytes. The maximum MAC frame size is 64 bytes. Figure 1 provides a summary of the basic Z-Wave frame structure.

Any certified Z-Wave product has a Z-Wave chipset onboard for interoperability with other devices from different manufacturers, as well as for backward compatibility with earlier versions. Since 2002, Z-Wave chipsets have evolved in terms of enhanced data rate, LR, low power, high performance, and encryption support.

B. NETWORK SECURITY FEATURES OF Z-WAVE

Because IoT devices communicate using radio frequency signals, their security is challenging as anyone in the vicinity can record the signal, and either replay it or spoof it back to the network. Considering the above-mentioned requirements, Z-Wave implements several security schemes for communication between the controller and endpoint devices. The Z-Wave transport encapsulation command class control [50] is defined as follows:

1) UNENCRYPTED COMMUNICATION USING CS-8 OR CRC-16 ENCAPSULATION

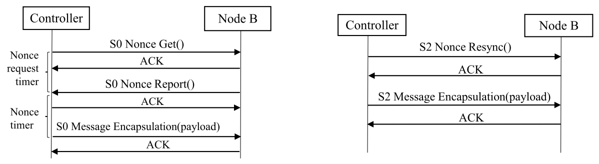
All controllers use this to communicate not only with legacy devices without encryption support owing to backward compatibility, but also with secure devices for network management traffic. In this mode, an additional CS-8 or cyclic redundancy check (CRC-16) checksum is added to validate the payload integrity of a frame. CS-8 and CRC-16 transport encapsulation is considered a non-secure communication, because it is vulnerable to a replay attack.

2) ENCRYPTED COMMUNICATION USING SECURITY 0 (S0)

This mode is used for secure application communication between S0 authenticated Z-Wave devices such as door locks, garage door remote openers, and the controller. The main goal of S0 is to ensure confidentiality, authentication, and replay attack prevention of the Z-Wave application layer payload using advanced encryption standard (AES-128) [51] encryption. Figure 2a illustrates the sending of a single S0 message that requires three commands and three ACKs, resulting in increased device power consumption [50]. The S0 network key is shared among all Z-Wave devices; therefore, a man-in-the-middle (MiTM) attack can retrieve it during the initial device network inclusion [5].

3) ENCRYPTED COMMUNICATION USING SECURITY 2 (S2)

S2 is the newest security class that uses AES-128-CCM [52] for encryption and authentication, and the elliptic curve Diffie Hellman (ECDH) [53] for secure network key derivation to alleviate S0 network key vulnerability [5]. S2 reinforces the confidentiality, authentication, and integrity of Z-Wave device communications. Figure 2b illustrates an S2 secure message, resulting in a lower device power consumption. The latest devices that support S2 are considered to be secured; however, they must also support unencrypted CS-8, CRC-16, and S0 communication because of the Z-Wave mandatory backward compatibility and for network management purposes. Consequently, the S2 device was downgraded to a weaker S0 security scheme during the initial device network inclusion [54]. Moreover, Section V presents the attack vectors on the S2 devices.



(a) Sending S0 secure message (b) Sending S2 secure message

FIGURE 2. **S0 and S2 message encapsulation.**

C. THREAT MODEL AND MOTIVATION

Owing to the great demand for IoT smart home devices, manufacturers usually rapidly release new devices with a focus on new functionalities rather than executing adequate security testing; thus, resulting in security breaches.

1) SMART HOME SECURITY THREATS

Smart home systems increase the security complexity as they encompass several layers: device, controller, cloud, and mobile application. All these layers, if not well implemented, could create opportunities for attacks on devices because each one uses different technology and communication protocols that could lead to security breaches, protocol specification violations, and logical faults in automation apps.

Moreover, these weaknesses can steer several bugs in smart home devices such as timing faults, sensor blinding, improper handling of timeout, faults in handling exceptional cases, weak device authentication, state confusion, fake and missing events, device remote control, over-privileged capabilities in mobile apps, device state out of synchronization, unexpected trigger action, denial of execution (DoE), and DoS [55], [56].

2) ATTACKER GOALS

With the above-mentioned security risks in home automation systems, an attacker might wish to have access to the smart home to either steal valuable data of the house owner or to conduct criminal offenses. With adequate skills and tools, the attacker might control, disable, or masquerade smart home devices, and have access to the house, while siren devices are not triggered, and online notifications are not sent to the cloud. Hence, preventing the house owner from being notified of the intrusion through his mobile app as demonstrated in Section V-E.

Also, the attacker could gain access to the device logs to obtain the daily usage pattern and confidential information of the house owner that could be exploited or sold to marketing companies. The attacker can manipulate vulnerable devices at their will and deny any communication to harass the house owner, to avoid triggering alerts, to avoid leaving a trace, or claim paid repair service. One of the worst cases will be to remotely turn on a smart gas controller and manipulate high-energy-consuming devices connected to smart switches, such as a stove and an electric heater, which could cause damage to the house and increase the energy bills.

3) Z-WAVE DEVICES ATTACK VECTOR

As Z-Wave devices operate wirelessly, an attacker could be near the house and sniff, with adequate equipment, the Z-Wave smart home network traffic. The attacker could retrieve the network information to either jam the device communication or inject malicious traffic that causes the Z-Wave devices to malfunction. As illustrated in Section I, unencrypted and S0 devices are widely used in smart homes and constitute the majority of products sold until late 2018, as the Z-Wave Alliance required device manufacturers to support S2 security on new devices only from April 2017 onward [57]; thus, leaving early adopting smart home users at risk.

Moreover, in Subsection V-E, we provide a real attack scenario on Z-Wave devices using a portable Raspberry Pi. The vulnerabilities found by VFuzz helped create exploits that could make the smart home controller services completely obsolete. With this simple implementation, an attacker could have easy access to the smart home and misuse devices at their will without leaving any trace.

4) MOTIVATION

In consideration of the above-mentioned attacks, there is a need to efficiently assess devices before release because it is difficult to patch IoT devices after deployment, and legacy Z-Wave devices are one-time-programmable (OTP) and cannot be updated. Therefore, the goal of our research is to conduct a security test on Z-Wave devices that are used in an actual smart home to help not only the end consumer discover the potential security flaws of the devices but also the manufacturers fix them. As we intend to use fuzzing to assess Z-Wave device vulnerabilities, we discuss our methodology next.

IV. METHODOLOGY

We aim to provide an easy-to-use fuzzer targeting Z-Wave devices (*e.g.*, a wide variety of modern smart-home devices) to not only find exploitable vulnerabilities, but also allow security professionals and end-users to assess potential threats to their devices. In this section, we present the challenges of systematically testing Z-Wave devices and the methodology used to develop VFuzz to deal with these challenges.

A. CHALLENGES OF Z-WAVE FUZZING

Fuzzing is a decades-old technique that is widely used for testing software programs, and its practical merits have already been proven with several new software bugs revealed. Moreover, several approaches have demonstrated that even IoT protocols such as 6LoWPAN, ZigBee, and Bluetooth Low Energy (BLE), can be assessed via fuzzing [31]–[35]. However, despite its widespread use in practice, systematic fuzz testing of the Z-Wave protocol has not yet been studied. Unlike other protocols, it is a proprietary protocol whose source code is not open-sourced, and the implementation is only available through Z-Wave SoC (System on Chip), which is completely blackbox; thus, it is challenging to debug.

Specifically, applying the fuzzing technique to an entirely new context of Z-Wave devices poses a unique set of new challenges, such as mutating structured packet data, remotely testing physical devices, and capturing bugs and state transitions from blackbox devices.

1) CHALLENGE 1: MUTATING SEMANTICALLY STRUCTURED DATA

Z-Wave packet frames are not only structured like any other network protocol, but also carry hierarchically organized information to describe the semantics (*i.e.*, functions and commands) per device type. Most traditional fuzzers (e.g., AFL and libFuzzer [58]) are agnostic to the structure of the data when applying mutation operators, and existing IoT fuzzers targeting other protocols cannot efficiently oversee Z-Wave specific semantics, which makes it challenging to directly use them when dealing with Z-Wave packets.

Solution 1. Using the domain knowledge that we accumulated from digesting the Z-Wave protocol specification, we designed a mutation logic called FIPA (**Fi**eld **P**rioritization **A**lgorithm), which is aware of the structure as well as the semantics of a packet frame; thus, making VFuzz capable of not only generating valid packets, but also actively investigating and inferring the device status.

2) CHALLENGE 2: REMOTELY TESTING PHYSICAL DEVICES

Unlike regular software programs that we can execute, communicate with, and test locally, Z-Wave devices are ''real,'' and they reside in the networks. In other words, we do not naturally obtain full control over the devices under test, and reasonable ways to locate devices, transmit mutated data, capture responses, or even switch them on/off need to be newly devised for VFuzz to be operational.

Solution 2. As a crucial element of VFuzz, we propose, implement, and build a cyber-physical system that includes both software and physical components as artifacts of VFuzz, which enables us to reliably assess network devices over-the-air.

3) CHALLENGE 3: CAPTURING BUGS AND FEEDBACK FROM BLACKBOX DEVICES

As our target is a physical network device, capturing its states is extremely challenging; a device could internally make a transition to an illegitimate state or crash silently without the fuzzer knowing about it. Nonetheless, it is essential for a fuzzer to use these states to detect bugs and generate feedback for efficient exploration of the input space. Certain fuzzing approaches that consider similar settings [38] rely on emulators to deal with such blackbox systems. However, no emulators for Z-Wave devices exist to date, and developing an emulator itself is not feasible as neither the Z-Wave protocol implementation nor the device implementation is open-sourced.

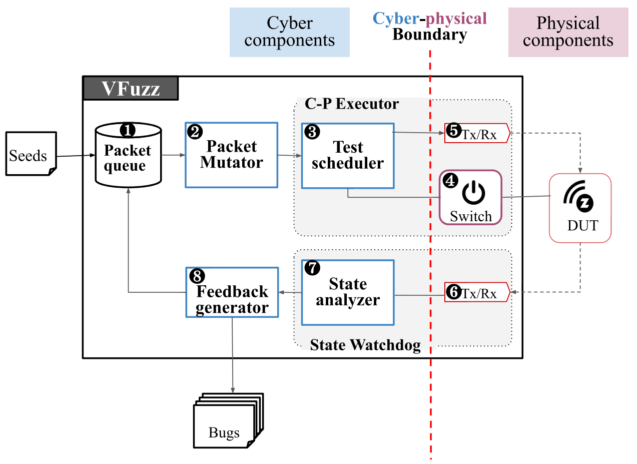


FIGURE 3. Overview of VFuzz's components and workflow.

Solution 3. We categorized a list of assorted Z-Wave devices into classes using their characteristics (*e.g.*, device type) as criteria, and build oracles for each class by mapping key indicators with device states. By observing the indicators listed in the mapping, VFuzz could determine issues in devices or generate feedback to the packet mutator.

B. OVERVIEW

With the aforementioned challenges in mind, we designed VFuzz, a mutation-based Z-Wave fuzzing framework that combines both cyber and physical components to effectively assess Z-Wave devices. Figure 3 illustrates the components and workflow of VFuzz. Starting from a valid Z-Wave packet as a seed (①), VFuzz's packet mutator (②, Subsection IV-C) applies a protocol-aware mutation scheme, called *FIPA*, which generates syntactically valid yet potentially malformed Z-Wave packets that could induce unexpected state transitions of the device under test (DUT).

VFuzz's cyber-physical executor (Subsection IV-D) orchestrates the overall execution of a fuzzing round (③) by turning on the target DUT via a smart switch device (④), establishing a connection between VFuzz and the device, and feeding the mutated packet over the air through a compatible Z-Wave dongle (⑤).

The state watchdog (Subsection IV-E) includes a Z-Wave transceiver (⑥) that monitors the DUT to capture any response and state transition whenever a mutated packet is transmitted. The captured state transitions and the response (or the absence of response) of the device cross the cyber-physical boundary again and are assembled by the state analyzer (⑦) to be handed over to the feedback generator (⑧, Subsection IV-F). Receiving the multiplexed state, VFuzz's feedback generator enqueues the mutated packet if the execution feedback is interesting and generates a report if any bug is detected.

C. PACKET MUTATOR

As Z-Wave devices operate wirelessly by sending and receiving signals over the radio frequency band, the input space is

F	M_T	M_T	D	F	M_T	M_A	M_A	M_A	D
H-ID	SRC	FC	LEN	DST	HDR	CmdCL	CMD	PARAM	CS

FIGURE 4. Classification of the fields constituting the packet frame of the current Z-Wave protocol, based on the proposed packet model.

limited to the packets conforming to the Z-Wave protocol described in Section III. Accordingly, VFuzz creates and mutates Z-Wave packets and transmits them to the DUT. By default, the maximum size of the transport layer frame of Z-Wave is 64 bytes, and as noted in a previous study [43], it takes $4 * 10^{146}$ years to assess each possible Z-Wave frame at the rate of sending one frame per second to the target device. Thus, for an effective and feasible testing, we need to carefully devise an optimized mutation strategy in terms of the fields that must be mutated, the mutation operator that should be applied to them, and proper schedule regarding when to mutate them.

1) Z-WAVE PACKET MODEL

Many mutation strategies could be set up by leveraging the domain knowledge of the Z-Wave specification. To reduce the size of the input space while keeping the chances for a packet to be accepted and trigger bugs high, a Z-Wave packet Z could be modeled as a union of four disjoint sets, considering the role of the field:

$$Z = F \cup D \cup M_T \cup M_A \tag{1}$$

where

- F is the set of *fixed fields*. The values are fixed per target device, and VFuzz never mutates them, *e.g.*, the home ID and device ID.
- D is the set of *dependent fields*. The values are determined absolutely by the values assigned to the rest of the fields, *e.g.*, checksum.
- M_T is the set of *mutable transport fields*. The values decide how the packet will be transported through the Z-Wave mesh network, *e.g.*, by setting intermediate nodes to the destination device in a routed packet.
- M_A is the set of *mutable application fields*. These values describe the functions of a device and how they should be processed.

The packet model was devised by studying the past and current Z-Wave specifications. Considering the mandatory backward compatibility policy of Z-Wave and device interoperability requirements, it is likely that future specifications also fit into our generic model. Each field of the current Z-Wave frame structure could be classified accordingly (see Figure 4).

- $F = \{\texttt{H-ID}, \texttt{DST}\}$. These fields are strictly fixed because they define the target device and the Z-Wave network to which it belongs.
- $D = \{\texttt{LEN}, \texttt{CS}\}$. The `LEN` is determined by the length of the packet, and `CS` is determined by the checksum. If the device properly implements the Z-Wave protocol, mutating either field would likely get the packet rejected

by the target. However, unlike the fixed fields, we should not completely exclude these fields from mutation candidates, because when dealing with protocols, we could expect multiple erroneous cases such as protocol not being implemented correctly missing checksum verification, procedures being under-implemented and failing to serve expected functions, or more than documented functions are implemented, *e.g.*, through code cloning, where mutated dependent fields could be interpreted differently than the original purpose. These are sufficient to render unsafe environment for users, and the packet mutator of VFuzz covers such scenarios by occasionally mutating the dependent fields.

- $M_T = \{\mathtt{SRC}, \mathtt{FC}, \mathtt{HDR}\}$. SRC specifies the node from which the packet is originates, and FC controls the transport frame type, such as ACK, singlecast, multicast, and routed frames with additional properties defined in the HDR field. Mutating these fields with random values could trigger bugs that are related to either bad protocol implementation or network communication errors. For instance, sending a packet where the SRC is equal to the DST with invalid network routing information could trigger improper handling of timeout and hop count, and invalid update of the target device routing table. Also, it could be interesting to observe the device response to the malformed packets.
- $M_A = \{\mathtt{CmdCL}, \mathtt{CMD}, \mathtt{PARAM}\}$. According to the Z-Wave protocol specification, mutable fields could be hierarchically classified into root fields (Command class), intermediate fields (Command), and terminal fields (Parameters), where the value set to the parent field determines the legitimate values of the child field. For instance, Table 1 presents the hierarchy regarding the "Binary Switch" Command Class registered to a smart switch device. The Binary Switch defines three child commands to *set* the state of the switch, to request the current state of the switch (*get*), and for the switch to *report* its current state. The specific action is determined using the parameter value. For example, to set the switch on, a controller should set CmdCL, CMD, and PARAM fields to "Binary Switch", "Set", and any value in the range 0×01–0×63 or 0xFF, respectively. Note that Z-Wave documentation explicitly states that commands with invalid parameter values, that is in the range 0×64–0xFE, had to be rejected. Therefore, the mutation of these fields is designed to cover both legal and illegal values, while also assessing the correctness of the device internal protocol implementation.

2) MUTATION OPERATORS

An effective fuzzer must ensure that all inputs in the input space should be reachable through mutations. As listed in Table 2, VFuzz features the operators that fully use the semantics of the fields, as well as those adopted from classic mutation-based fuzzers as they are proven to be effective in generating critical values for exploring various program behaviors. These mutation operators are coupled with the types of fields, as summarized in Table 3. Fixed fields do not have mutation operators as they might not be mutated. Meanwhile, each field belonging to the mutable field has a predefined set of semantically valid and invalid values. For example, when mutating the PARAM field of a Binary Switch Set packet shown in Table 1, rand_valid randomly chooses a value from $\{0 \times 00, 0 \times 01, \ldots, 0 \times 63$, 0xFF$\}$, which would either turn the switch on or off, and rand_invalid selects a random value in the range 0×64–0xFE, that the DUT should ignore, as per specification.

TABLE 1. An example of the hierarchy of the mutable application fields of a Command Class Binary Switch associated with a smart switch device, whose basic function is to turn the power switch on or off.

Root CmdCL	Intermediate CMD	Terminal PARAM	
0x25 (Binary Switch)	0x01 (Set)	0x00	(Off)
		0x01–0x63	(On)
		0xFF	(On)
		0x64–0xFE	(Invalid)
	0x02 (Get)	-	-
	0x03 (Report)	0x00	(Off)
		0xFE	(Unknown)
		0xFF	(On)
		0x01–0xFD	(Invalid)

TABLE 2. List of mutation operators VFuzz uses. Bitflip, byteflip, arithmetic, interesting, and insert operators are adopted from the AFL fuzzer.

Operator	Description
rand_valid	Replace with a randomly selected legal value.
rand_invalid	Replace with a randomly selected illegal value.
bitflip	Flip a bit at a random position.
byteflip	Flip a byte at a random position.
arith	Add/subtract small integer.
interesting	Replace with interesting values.
insert	Append a random byte.

3) MUTATION SCHEDULING

Given the mutation candidates and corresponding mutation operators, FIPA, as illustrated in Algorithm 1, schedules the mutation to efficiently explore the input space.

D. CYBER-PHYSICAL EXECUTOR

The cyber-physical executor of VFuzz bridges the cyber components with the physical components by executing the DUT through a software-controllable switch, establishing a connection with the device, and sending the mutated packet to the device via TX modules.

1) DEVICE POWER MANAGEMENT, CONNECTION, AND PROBING

The Switch, as illustrated in Figure 3, ensures that the device is restarted in the case of a crash or bug. For instance,

TABLE 3. Mutation operators assigned to each field. F, M, and D in Type refer to fixed, mutable, and dependent fields, respectively.

Field	Type	Len	Mutation operators
H-ID	F	4	None
DST	F	1	None
SRC	M_T	1	rand_valid, rand_invalid
FC	M_T	2	rand_valid, rand_invalid
SQ	M_T	1	rand_valid, rand_invalid
HDR	M_T	0+	rand_valid, bitflip, byteflip, arith, interesting, insert
CmdCL	M_A	1	rand_valid, rand_invalid
CMD	M_A	1	rand_valid, rand_invalid
PARAM	M_A	0+	rand_valid, bitflip, byteflip, arith, interesting, insert
LEN	D	1	bitflip, byteflip, arith, interesting, insert
CS	D	1	bitflip, byteflip, arith, interesting, insert

a Z-Wave smart power strip could be used to automatically turn on/off the power of a target smart wall plug connected to it. Once the device is powered on, VFuzz first tests whether it is responsive by sending a Z-Wave NO_OPERATION (NOP) frame, which is like the TCP/IP ICMP Echo. VFuzz uses the NOP frame to query and probe the status of the target device. If the target device responds with an ACK, the connection is considered established, and VFuzz sends the mutated packet. Figure 5 shows the VFuzz message flow.

2) DEVICE CAPABILITIES RETRIEVAL

For an effective packet mutation, VFuzz needs to know which CmdCL the target device supports to prioritize their mutation and learn its encryption capabilities. This is achieved by sending a NODE INFORMATION (NIF) GET frame to the target device, which would respond with its NIF REPORT frame listing all its capabilities and supported CmdCL. After receiving the NIF REPORT, VFuzz prioritizes the mutation of supported CmdCL and analyzes the device encryption support by searching in the NIF REPORT packet for values 0×98 and $0 \times 9F$, which correspond to S0 and S2 encryption supports. If the target device does not support encryption, it is vulnerable to packet injection, replay attacks, impersonation, and remote control.

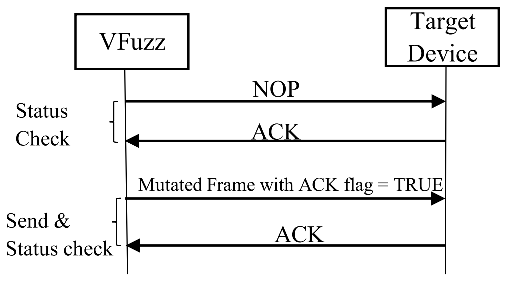


FIGURE 5. Initial message flow between VFuzz and target device.

3) SENDING PACKET OVER THE AIR VIA TX/RX MODULES

The TX/RX modules provide a configuration that allows the fuzzer to send and receive packets via SDR devices or supported dongles. These modules use third-party libraries

Algorithm 1 Field Prioritization Algorithms (FIPA)

```
Input  : S: set of seed Z-Wave packets,
         T: target Z-Wave device under test,
         O: set of bug oracles,
         H-ID: unique identifier of Z-Wave network,
         DST: unique identifier of T,
Output : B: set of packets that trigged bugs
procedure VFuzz(H-ID, DST, T)
    /* Main module of VFuzz */
    queue ← S;
    while True do
        p ← queue.dequeue()
        p′ ← Mutate(p, H-ID, DST)
        if TestAndCheck(p′, T, O) == True then
            B ← B ∪ {p′}
            /* A mutated packet is flagged as Interesting if
               it increases T response time. */
        else if isInteresting(p′, T) == True then
            queue.enqueue(p′)
    if queue == ∅ then
        queue ← S
    return B
procedure TestAndCheck(p′, T, O)
    /* Tests device T with the mutated packet p', and
       checks the device states against oracle O. */
    Test(T, p')
    isBug ← Check(T, O)
    return isBug
procedure Mutate(p, H-ID, DST)
    /* Packet p has 10 fields:
       H-ID,SRC,FC,LEN,DST,HDR,CmdCL,CMD,PARAM,CS. Each
       field has a name, type, and value. */
    p[0].value ← H-ID
    p[4].value ← DST
    /* Choose and mutate a mutable transport or mutable
       application field. */
    mut_fields ← {p[1], p[2], p[5], p[6], p[7], p[8]}
    field ← random_choice(mut_fields)
    ops ← get_all_mut_operators(field)
    op ← random_choice(ops)
    p[field.index].value ← op(field.name, field.value)
    /* Properly set dependent fields. */
    p[3].value ← updateLength(p)
    p[9].value ← updateChecksum(p)
    /* Occasionally choose and mutate a dependent field.
       */
    if random() < mut_dep_prob then
        dep_fields ← {p[3], p[9]}
        field ← random_choice(dep_fields)
        ops ← get_all_mut_operators(field)
        op ← random_choice(ops)
        p[field.index].value ← op(field.name, field.value)
    return p
```

for packet management and radio processing, such as Scapy-Radio and GNU Radio for HackRF One SDR [59], and RFlib for YardStick One dongle [60].

VFuzz sends a mutated frame that contains an ACK request flag set to True; therefore, if the target processes the mutated frame, it has an obligation to send back an ACK. After receiving the ACK receipt, or not receiving it for a timeout period, VFuzz turns to the state watchdog to analyze the bugs or state transitions.

TABLE 4. Description of used oracles and corresponding bugs.

Oracle Type	Bug
`oracle_response_timeout`	DoS
`oracle_power_loss`	Crash
`oracle_uncontrolled_ressource`	Power bug
`oracle_overheating`	Safety bug
`oracle_invalid_sequence`	Specification violation bug
`oracle_switch_set`	Specification violation bug

E. STATE WATCHDOG

As discussed in Subsection IV-A, it is not trivial to infer the internal states of a Z-Wave device, which is a blackbox residing in a network. For example, when trying to check if the light bulb device quickly processes the switch on command and switches the light On, one might send an additional `GET` packet asking the bulb to report the status of the light. However, there are several scenarios that make such approach undesirable: (1) the bulb does not receive the GET packet at all, and could not report the light's state, (2) the bulb internally updates the light state to On, reporting back that the light is on, but fails to physically turn the light on, (3) the bulb physically turns the light on, but fails to update the light's state, reporting back that the light is off, and so forth. Thus, we propose a state watchdog that uses TX/RX modules to sniff the response received from the target device to monitor its approximate internal states as well as capture and validate its buggy states.

1) CHECKING BUGS WITH ORACLES

The goal of VFuzz is to detect bugs in Z-Wave devices, including those described in Subsection III-C. For each class of bugs and vulnerabilities, we define an oracle that the state analyzer could use to check for the existence of a bug. Table 4 lists some of the oracles used and corresponding bugs. Figure 6 shows a sample of the pseudocode related to the oracle that checks a specific Z-Wave Command Class `BINARY_SWITCH` with Command `SET`. The Z-Wave specification features 117 Command Classes and corresponding Commands with defined values. Note that in this paper, we initially focus on the Command Classes supported by our testbed devices, and plan to extend the oracles to include the rest of the commands in the future.

F. FEEDBACK GENERATOR

The feedback generator stores logs of all sent packets and received ones.

1) BUG REPORTS

Packets that cause bugs on the target device are logged separately for further user analysis and validation.

2) INTERESTING STATE TRANSITIONS

If no potential bug is detected, VFuzz checks if the device has made any state transition, which implies that the device is adversely affected by the mutated packet, even though it has not exhibited any buggy state. For example, if the response time continues to increase upon receiving mutated packets, it is reasonable to assume that further mutation on the packet will grant VFuzz a higher chance of triggering a DoS than sending a random packet. Based on this logic, we established a mapping between observable state transitions and their ''interesting-ness''. If any of the interesting state transitions are captured, the feedback generator enqueues the mutated packet in the packet queue so that the packet mutator continue to mutate the interesting packet and trigger a potential crash or bug.

```
def oracle_switch_set(p, pre_light_state,
    post_light_state):
  if (p.CmdCL == CMDCL_SWITCH_BINARY and p.cmd == SET):
    val = p.param
    if (val >= 0x01 and val <= 0x63) or val == 0xFF:
      # Turning light on
      correct_light_state = 1
      if (post_light_state != correct_light_state):
        report_bug()
    elif (val == 0x00):
      # Turning light off
      correct_light_state = 0  # Turn off
      if (post_light_state != correct_light_state):
        report_bug()
    else:
      # Illegal command
      if (pre_light_state != post_light_state):
        report_bug()
      if (device_responded_to_p):
        report_bug()
```

FIGURE 6. Pseudocode of the correctness oracle checking the Binary Switch Set Command.

V. EVALUATION

We evaluated the impact of VFuzz by fuzzing various Z-Wave smart home devices. Specifically, we showed the overall effectiveness of VFuzz through the previously unknown vulnerabilities discovered (Subsection V-B), and the efficiency of the input mutator (Subsection V-C) as well as the cyber-physical executor (Subsection V-D) compared to the relevant work. Also, we provided a proof-of-concept attack scenarios to demonstrate the practical impact of the vulnerabilities found by VFuzz on a real Z-Wave smart home (Subsection V-E).

A. EXPERIMENTAL SETUP

1) TARGET DEVICES

To assess diverse versions and implementations of the Z-Wave protocol, we set up a Z-Wave testbed that consists of both the latest S2, S0, and legacy devices from several manufacturers. Out of all the Z-Wave chipset series (100, 200, 300, 400, 500, and 700), we could find only devices from series 300, 500, and 700 on the market, because of obsolescence by defects in the 200 and 400 chipsets [61]. Figure 7 shows the devices used in this study, and Table 5 lists their specifications.

2) BASELINE FUZZERS FOR COMPARISON

To the best of our knowledge, there is no systematic fuzzing approach targeting Z-Wave devices available, other than

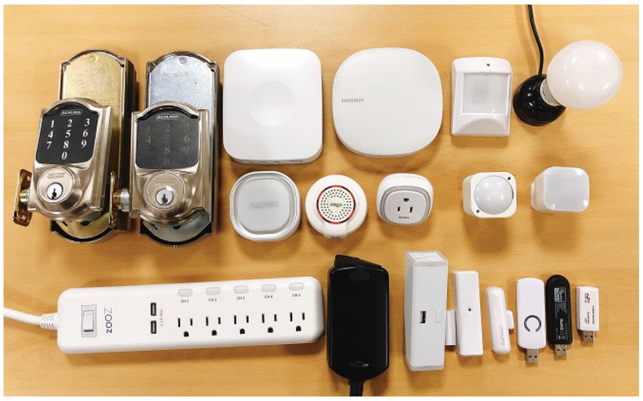


FIGURE 7. Z-Wave devices used for experiments. Clockwise from the top left: Schlage door lock BE468, Schlage door lock BE469ZP, Samsung SmartThings Hub, Samsung ConnectHome, Linear Motion Sensor, Linear smart LED bulb, Aeotec ZWA008 motion sensor, Aeotec ZW100-A motion sensor, Fibaro wall plug, Dome siren, Aeotec siren, SiLabs UZB-7 controller, Zooz S2 USB Stick, Aeon Labs Z-Stick Gen 5, Aeotec contact sensor, Linear contact sensor, Zooz S2 double plug with USB, Jasco Plug-in smart switch, and Zooz power strip.

H-ID	SRC	FC	SQ	LEN	DST	HDR	CmdCL	CMD	PARAM	CS
0xC6BD818E	0x01	0x41	0x01	0x0E	0x1A	0x00	0x00	0x00	0x00	0xDE

FIGURE 8. Initial seed example with added `H-ID` and `DST` .

a work close to VFuzz, i.e., EZ-Wave [7]. The authors of EZ-Wave identified one vulnerability in one Z-Wave device by sending crafted Z-Wave frames. However, as its algorithm for packet generation is not known, we could only perform a result-oriented comparison with EZ-Wave regarding the throughput and bugs found. Also, to evaluate the effectiveness of packet mutation, we compared FIPA to Radamsa [62] and a random algorithm by implementing their mutation modules in customized versions of VFuzz.

3) INITIAL SEED SETS

Unless otherwise specified, the initial seed packet is a basic `NOP` packet with `H-ID` and `DST` fields properly set to those of the testbed network and the target device respectively, and `LEN` and `CS` set to correct length and checksum values (see Figure 8).

4) EXPERIMENT ENVIRONMENT

We evaluated VFuzz on a desktop machine with an Intel Core i5-6600 (3.3 GHz), 8 GB RAM, and 256 GB SSD, running Ubuntu 18.04 as the host OS. An earlier prototype version used HackRF One [59] as the transceiver; however owing to the short transmission range and high cost, we opted for the YardStick One [60] dongle. The selection of YardStick One was based solely on its low price, small size, capabilities, and transmission range. VFuzz ranges from 20 m indoors to 30 m outdoors.

5) EVALUATION METRICS

Evaluating blackbox fuzzers on IoT devices is challenging because of the lack of source code and memory debug analysis tools used in whitebox and greybox fuzzing for assessing code coverage [30], [63]. To assess VFuzz, we suggest three metrics: vulnerability discovery, mutator efficiency, and executor efficiency.

- **Metric 1: Vulnerability discovery.** Measures the effectiveness of VFuzz in finding vulnerabilities in real Z-Wave devices. For crash triage, we manually assess packets to remove redundancy and identify unique exploitable vulnerabilities.
- **Metric 2: Mutator efficiency.** This metric measures the efficiency of FIPA in generating semi-valid Z-Wave packets and buggy packets, compared to other mutation algorithms. It accesses the average ratio between packets successfully received by the target device to the total number of packets sent by VFuzz.

$$\textit{Reception Ratio} = \frac{\textit{Total Packets Received}}{\textit{Total Packets Sent}} \tag{2}$$

- **Metric 3: Executor efficiency.** It determines VFuzz input generation speed or throughput per second in consideration of the processing power of the target device.

$$\textit{Throughput} = \frac{\textit{Total Packets Generated}}{\textit{Time (sec)}} \tag{3}$$

B. VULNERABILITY DISCOVERY

First, the previously unknown vulnerabilities discovered by VFuzz are listed. Because of fuzzing the devices listed in Table 5, VFuzz discovered 10 critical vulnerabilities (see Table 6), causing devices to either malfunction, crash, or become unresponsive, or let the attacker control the device without authentication. From these flaws, six CVEs were assigned by the US CERT/CC (see Table 7). Below-mentioned are certain interesting cases.

1) CONTROLLER DoS

The most critical issue found in all five Z-Wave controllers (*i.e.*, D1 through D5) renders them vulnerable to DoS attacks, which make their service inaccessible. The buggy packet frame is generated because of mutating the `HDR` to 0×01 and the `CmdCL` to 0×04 `(FIND NODES IN RANGE)` with a randomly inserted payload. This CmdCL causes the controller to search for neighbors' devices. As the payload is random, the controller searches for rogue devices by continuously sending `NOP` and could not proceed with any upcoming device events. Thus, the user could not be notified about any of the events happening in the smart home, such as burglary, fire alarm, water leak, and gas leak. The bug is due to the Z-Wave specification, the lack of authentication of the sender, and the lack of verification of the packet's application payload.

2) CONTROLLER CRASH

This vulnerability causes the controller to crash and requires a power reset. This occured when VFuzz mutated the Z-Wave frame control field with a valid value of the route property set to `True` and invalid `HDR`, `CmdCL`, `CMD`, and `PARAM`.

TABLE 5. Detailed specifications of the tested Z-Wave devices regarding the type, vendor, model number, chipset, firmware, supported encapsulation mode, supported encryption mode, and the power source. We will refer to each device as the identifier specified in the **ID** column herein.

Device Type	ID	Vendor	Model (Year)	Chipset	Firmware	Encapsulation	Encryption Support	Power Supply
Controller	D1	SiLabs	UZB-7 (2019)	700	7.00	CRC-16, S0, S2	AES-128, ECDH	USB 5V
	D2	Zooz	ZST10 (2018)	500	6.04	CRC-16, S0, S2	AES-128, ECDH	USB 5V
	D3	Aeon Labs	ZW090-A (2018)	500	3.95	CRC-16, S0, S2	AES-128, ECDH	USB 5V
	D4	Samsung	ET-WV520 (2017)	500	6.04	CRC-16, S0, S2	AES-128, ECDH	AC 120V
	D5	Samsung	STH-ETH-200 (2015)	500	6.04	CRC-16, S0	AES-128	AC 120V
Door Lock	D6	Schlage	BE469ZP (2019)	500	6.03	CRC-16, S0, S2	AES-128, ECDH	Battery 6V
	D7	Schlage	BE468 (2014)	500	3.42	CRC-16, S0	AES-128	Battery 6V
Contact Sensor	D8	Aeotec	ZWA008 (2019)	500	6.07	CRC-16, S0, S2	AES-128, ECDH	Battery 3.6V
	D9	Linear	WADWAZ-1 (2015)	300	3.43	CRC-16	No	Battery 3V
Motion Sensor	D10	Aeotec	ZWA005-A (2019)	500	4.61	CRC-16, S0, S2	AES-128, ECDH	Battery 3V
	D11	Aeotec	ZW100-A (2017)	500	4.05	CRC-16	No	Battery 6V
	D12	Linear	WAPIRZ-1 (2015)	300	3.43	CRC-16	No	Battery 3V
Siren	D13	Aeotec	ZW164-A (2019)	500	5.03	CRC-16, S0, S2	AES-128, ECDH	AC 120V
	D14	Dome	DM501 (2016)	500	4.26	CRC-16	No	Battery 6V
Power Strip	D15	Zooz	ZEN20 (2019)	500	5.03	CRC-16, S2	AES-128, ECDH	AC 120V
Smart Switch	D16	Zooz	ZEN25 (2018)	500	5.03	CRC-16, S2	AES-128, ECDH	AC 120V
	D17	Fibaro	FGWPB-111 (2018)	500	4.3	CRC-16, S0, S2	AES-128, ECDH	AC 120V
	D18	Jasco	ZW4201 (2016)	500	4.05	CRC-16	No	AC 120V
LED Bulb	D19	Linear	LB60Z-1 (2014)	500	3.5	CRC-16	No	AC 120V

TABLE 6. Summary of the new vulnerabilities detected by VFuzz, devices affected by each vulnerability, and the average time required to trigger the vulnerability across five runs of fuzz testing.

N	Vulnerability Type	Affected devices	Time
1	Controller DoS	D1-D5	1 h 45 s
2	Controller Crash	D1-D3	2 h 50 m
3	Device Crash	D15-D17	2 h 45 m
4	Door Lock DoS 1	D7	1 h 30 m
5	Door Lock DoS 2	D7	72 h 3 m
6	Device Remote Control	D14, D18, D19	45 s
7	Lack of Data Authenticity	D1-D7, D14-D19	11 s
8	Information Disclosure	D1-D7, D14-D19	23 s
9	Information Tampering	D1-D7, D14-D19	32 s
10	Specification Violation	D1-D19	50 s

These malicious frames corrupt the controller routing table. Moreover, it was identified by the state watchdog due to increase in target device response. It flags a crash when a target device cannot communicate for more than 1 min.

The Z-Wave routing protocol implements a maximum of four repeaters (four hops) to reach distant smart home devices. The above-mentioned vulnerability could affect the smart home responsiveness. Because an attacker could inject false routing hops information into a vulnerable controller's routing table, which could result in a delay in command processing time.

3) DEVICE CRASH

Devices D15, D16, and D17 crashed when VFuzz sent mutated packets with `CmdCL` 0×98 0×40 `(SECURITY NONCE GET)`, $0 \times$ `9F` 0×01 `(SECURITY 2 NONCE GET)`, 0×00 `(NO OPERATION or NOP)`, and 0×01 0×02 `(NIF REQUEST)`. These packets were flagged as "*interesting*" by the state watchdog because they caused the target device transmission delay and led to uncontrolled resource consumption vulnerabilities.

4) DOOR LOCK DoS 1

Despite the S0 security features of door lock D7, it accepts non-authenticated external packets from VFuzz. The DoS was reached while sending a mutated packet to the controller with an invalid `NIF` of device D7, which falsely reports the non-support of S0 encryption. After validating and updating the malicious `NIF` in its routing table, the controller no longer control the door lock. To recover from this attack, a factory reset of the door lock and manual network inclusion is required.

5) DOOR LOCK DoS 2

The DoS was reached while fuzzing the target device D7 with `CmdCL` 0×00 `(NOP)`, 0×01 0×02 `(NIF REQUEST)` , and 0×98 0×40 `(SECURITY NONCE GET)` packets. These packets request the target to respond with its `ACK`, `NIF REPORT`,and `NONCE REPORT` values for network communication. While fuzzing with these above-listed crafted frames, the device respond indefinitely, resulting in high battery consumption. The door lock automatically unlocks when it reaches low battery level. This bug is caused by the lack of implementation of response rate limiting and sender authentication.

6) DEVICE REMOTE CONTROL

As legacy CRC-16 devices do not implement encryption, a remote control attack could be launched by repetitively sending a frame that deactivates the devices for a period; thereby, denying the user requests for activation. VFuzz generates the corresponding frame from the data retrieved from the `NIF` report of the target device. For instance, sending a packet with a `CmdCL` 0×20 0×01 0×00 `(BASIC SET OFF)`, 0×25 0 × 01 0 × 00 `(SWITCH BINARY SET OFF)` or 0 × 26 0 × 01 0 × 00 `(SWITCH MULTILEVEL SET OFF)` to the CRC-16 devices, namely, D14, D18, and D19, turn them off. The devices would not be accessible during the attack, despite the home user attempting to turn them on.

7) ADDITIONAL VULNERABILITIES

Most devices do not verify the authenticity of the data sent by VFuzz. As a result, fake events could be injected into the controller, which could lead to the actuation of predefined scenes. Several devices send back packets without encryption, which could be captured, tampered, and replayed. All devices accept packets from invalid and non-specified frame values; which is a violation of the Z-Wave protocol specification.

8) NEW CVE ON Z-WAVE CHIPSET

After collecting the buggy packets that made the target devices either unresponsive or crash, we cross-checked all devices to see if the same packet adversely affected multiple devices. During the process, we discovered that certain vulnerabilities affect multiple devices that share the same Z-Wave chipset series. Such vulnerabilities were due to the issues in the chipsets rather than the individual devices, which potentially imply higher security impacts. For those, CVE numbers were issued, and Table 7 summarizes the new CVEs found per Z-Wave chipset series along with the matching CWEs [64] and affected devices.

C. MUTATOR EFFICIENCY

We compared the efficiency of FIPA in generating valid Z-Wave packets to Radamsa, and Random mutators. Here, we used customized VFuzz to generate packets with FIPA, Radamsa, and random functions. Radamsa and Random mutation failed to generate a meaningful number of semi-valid Z-Wave packets as they are agnostic to the syntax and semantics of the Z-Wave protocol. This justifies our challenges listed in Section IV, because IP-based protocol mutators could not be directly applied to IoT-based protocols, which have different constraints such as the dependency of a `CmdCL`, `CMD`, and `PARAM`. Figure 9 illustrates the average packet reception ratio of the target device for 10,000 packets generated by each method per test mode. In test Mode 2, we considered the above-mentioned constraints by automatically adding valid `LEN` and `CS` values, and restricting the mutation of invalid application payload with `LEN` greater than 64 bytes. With these constraints, Radamsa and Random mutations performed better.

D. EXECUTOR EFFICIENCY

VFuzz has a higher throughput than EZ-Wave; VFuzz provides an average throughput of 54 with respect to the processing power of the target device, while EZ-Wave offers one test case per second (see Figure 10). One of the reasons is that VFuzz only needs to send two packets per test case, while EZ-Wave sends four packets. Moreover, VFuzz is efficient in finding new vulnerabilities owing to its low implementation complexity and diverse devices included in the testbed. Table 8 compares VFuzz and EZ-Wave.

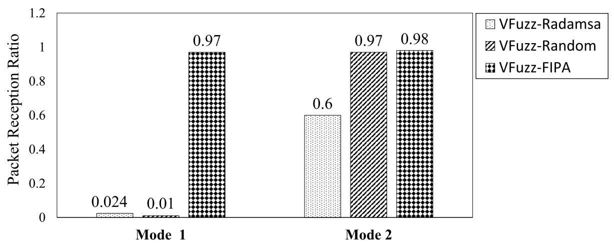


FIGURE 9. Average packet reception ratio (obtained in 5 runs) from Radamsa, Random, and FIPA mutation. Mode 1: mutation of whole Z-Wave initial seed packet. Mode 2: restricted mutation of the packet application payload only plus generation of valid `LEN` and `CS`.

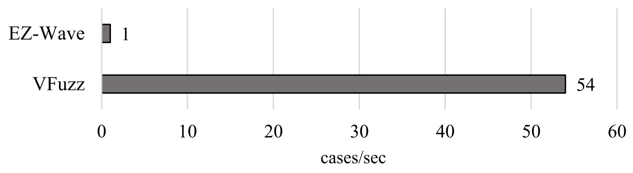


FIGURE 10. Comparison of the average throughput of EZ-Wave and VFuzz.

E. PROOF-OF-CONCEPT ATTACK SCENARIOS

With the knowledge of the discovered vulnerabilities during fuzz testing, we extended the VFuzz framework to include a proof-of-concept exploit mode (run as `VFuzz -e`) to attack a miniature Z-Wave smart home that resembles an actual smart home environment. This smart home runs a Z-Wave network that consists of the devices listed in Figure 7, namely a Z-Wave controller, door lock, windows contact sensor, smart LED, smart switch, and siren. For mobility and portability, VFuzz runs on a Raspberry Pi 4 separately from our Z-Wave testbed and injects Z-Wave packets directly to the targeted Z-Wave controller and slave devices. We provided a video of the attack in [28].

The attacker is located outside the smart home network and uses dongles connected to a Raspberry Pi 4 to sniff and record all the traffic of the Z-Wave smart home network. After capturing the packets, the attacker analyzes them and retrieves the `H-ID` and `SRC` of the targeted devices. After information retrieval, VFuzz generates new packets and injects them into the network. The attacker could wait until the house owner leaves, then inject commands to the Z-Wave network causing devices to either grant them access to the home, disable alarms, turn on high-power devices that consume energy, cause devices to malfunction, or cause DoS on the controller. The worst case would be to send repetitive commands that misuse, drain, and damage devices.

We performed a successful DoS attack on the controller that allows intruders to have physical access to the smart

TABLE 7. CVE IDs and vulnerabilities discovered per Z-Wave chipset.

Z-Wave Chipset	Firmware	Encryption Support	Affected Devices	Vulnerability with CWE ID	CVE ID	Fixable	Root Cause
100, 200, 300	3.43	None	D9, D12	CWE-74, 200, 294, 311, 345, 346	CVE-2020-9057	No	Specification
500	5.20	None	D14, D18, D19	CWE-74, 200, 294, 311, 345, 346	CVE-2020-9058	No	Specification
500	3.42	S0	D7	CWE-400, 346	CVE-2020-9059	Yes	Specification
500	6.04	S2	D2, D3, D15–D17	CWE-400, 346	CVE-2020-9060	Yes	Specification
500	6.04	S2	D1–D3, D5	CWE-285, 346	CVE-2020-9061	Yes	Specification
700	7.00	S2	D1	CWE-285, 346	CVE-2020-10137	Yes	Specification

CWE-74: Injection, CWE-200: Information Exposure, CWE-285: Improper Authorization, CWE-294: Authentication Bypass by Capture-replay, CWE-311: Missing encryption, CWE-345: Insufficient verification of Data Authenticity, CWE-346: Origin Validation Error, CWE-400: Uncontrolled Resource Consumption.

TABLE 8. Point-by-point comparison of VFuzz to EZ-Wave, a closely related testing approach targeting Z-Wave.

	EZ-Wave	VFUZZ
Technique	Pentest	Fuzzing
Z-Wave Compatibility	High	High
Throughput	Medium	High
Fuzzing Capabilities	Limited	Yes
1-day Vulnerabilities on CRC-16	Yes	Yes
0-day Vulnerabilities on S0	No	Yes
0-day Vulnerabilities on S2	No	Yes
Portable on Raspberry Pi 4	No	Yes
CVEs	0	6

home by disabling notifications from the contact sensor attached to the windows that detects a breach. Also, we performed a successful replay attack on the devices and successfully impersonated all devices by sending fake event packets to either the controller or other devices. As the controller does not check for the integrity of the sender, it accepts all the replayed packets and updates the device status to the user's mobile app via the cloud; thereby, providing incorrect information.

VI. DISCUSSION AND COUNTERMEASURES

In this section, we discuss the impact and limitations of VFuzz. In addition, we propose countermeasures.

A. SIMPLICITY

The direct interaction between VFuzz and the Z-Wave devices enables the fuzzer to achieve fast response and high efficiency, as VFuzz does not rely on a smartphone app of the devices or a cloud relay as compared to other IoT fuzzers that work on the controller IP interface or rely on middleware such as a mobile app [33].

B. RESEARCH SIGNIFICANCE

With more than 100 million devices on the market, the Z-Wave ecosystem faces several security challenges as illustrated in our research results in Section V. Various Z-Wave legacy CS-8 and CRC-16 devices are vulnerable to replay, injection, and DoS attacks because these devices do not implement encrypted communication and lack a replay protection mechanism. Moreover, certain legacy Z-Wave device chipsets are one-time-programmable and cannot be updated, *e.g.*, with a firmware update as with more recent S2 Z-Wave devices, to mitigate these attacks. Currently, only 27% of products implement S2 security that guarantees confidentiality, integrity, and authentication between Z-Wave devices [18]. However, S2 security needs to support specific unencrypted commands due to backward compatibility; consequently, VFuzz succeeded in injecting malicious packets causing a DoS.

C. LIMITATION ON S0 AND S2 DEVICES

Despite its benefits, VFuzz faces limitations that necessitate future enhancements. VFuzz has a high accuracy for unencrypted CRC-16 devices. For the S0 and S2 devices, we can only fuzz limited Z-Wave packets owing to the encryption requirement. There is a need to explore the possibility of fully fuzz encrypted S0 and S2 devices by retrieving the encryption key of a target Z-Wave network.

D. RECOMMENDATIONS

We recommend that Z-Wave device manufacturers implement S2 security by default on all new devices with a fix to downgrade attacks during device network inclusion [54]. In addition, a proper Z-Wave specification on S2 devices can eliminate the flooding attack by authenticating the sender and limiting the amount of received packets per device and per time frame. Moreover, if well patched, a Z-Wave firmware update can restrict malicious packets from being processed by S2 devices. For the legacy CRC-16 Z-Wave devices targeted in this paper, we suggest the development of an intrusion detection and prevention system either on the Z-Wave controller or a separate device to detect external attacks and alert the user for an immediate response. We advise smart home users to combine sensors using different technologies such as *e.g.*, Z-Wave and ZigBee, so that when the former is attacked, the latter could inform the user of an intrusion.

E. RESPONSIBLE DISCLOSURE

We filed several vulnerability reports to the US CERT/CC division [27] in order to work with the respective chipset and device manufacturers to fix and mitigate the threats that we discovered.

F. AVAILABILITY AND ETHICAL CONSIDERATIONS

The VFuzz public version provides core Z-Wave fuzzing functionalities to researchers while reducing advanced features that could be misused by bad actors to attack smart home devices. For the same ethical considerations, we are not releasing the VFuzz PoC exploit code.

The VFuzz public source code will be available for download at https://github.com/CNK2100/VFuzz-public.

VII. CONCLUSION AND FUTURE WORK

In this paper, we presented VFuzz, a Z-Wave fuzzing framework that checks for known and unknown vulnerabilities in Z-Wave smart home devices. Using the semantic-aware Field Prioritization Algorithm for mutation and feedback mechanism assisted by a robust state watchdog, VFuzz tested actual Z-Wave devices that have already been deployed to the market and found six new security CVEs (see Table 7) on Z-Wave chipsets including (1) critical vulnerabilities in Z-Wave legacy CRC-16 devices, and (2) specific attack vectors in secured S0 and S2 devices owing to backward compatibility and protocol specification requirements.

We suggest vendors to actively participate in patching the firmware and roll out updates to the affected devices to protect users from critical security issues. For future work, we are developing an intrusion detection and prevention system for Z-Wave legacy non-secure devices.

REFERENCES

[1] International Data Corporation. *Worldwide Internet of Things Forecast, 2020–2024*. Accessed: Dec. 14, 2021. [Online]. Available: https://www.idc.com/getdoc.jsp?containerId=US45861420

[2] Sigma Designs. *Sigma Designs Ships 100 Millionth Z-Wave Chipset Module*. Accessed: Dec. 14, 2021. [Online]. Available: https://www.globenewswire.com/news-release/2018/04/12/1469417/0/en/Sigm a-Designs-Ships-100-Millionth-Z-Wave-Chipset-Module.html

[3] Silicon Laboratories. *Smart Home Control on One App*. Accessed: Dec. 14, 2021. [Online]. Available: https://www.z-wave.com/

[4] Z-Wave Alliance. *The Smart Home is Powered by Z-Wave*. Accessed: Dec. 14, 2021. [Online]. Available: https://z-wavealliance.org.

[5] B. Fouladi and S. Ghanoun, ''Security evaluation of the Z-wave wireless protocol,'' *BlackHat USA*, vol. 24, pp. 1–2, Jan. 2013.

[6] 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 *Proc. 9th Int. Conf. Crit. Infrastruct. Protection (ICCIP)*, in Critical Infrastructure Protection IX, vol. 466, M. Rice and S. Shenoi, Eds. Arlington, VA, USA, Mar. 2015, pp. 209–227. [Online]. Available: https://hal.inria.fr/hal-01431003, doi: 10.1007/978-3-319-26567-4_13.

[7] J. Hall, B. Ramsey, M. Rice, and T. Lacey, ''Z-wave network reconnaissance and transceiver fingerprinting using software-defined radios,'' in *Proc. Int. Conf. Cyber Warfare Secur.*, 2016, p. 163.

[8] J. D. Fuller and B. W. Ramsey, ''Rogue Z-wave controllers: A persistent attack channel,'' in *Proc. IEEE 40th Local Comput. Netw. Conf. Workshops (LCN Workshops)*, Oct. 2015, pp. 734–741, doi: 10.1109/LCNW.2015.7365922.

[9] L. Rouch, J. François, F. Beck, and A. Lahmadi, ''A universal controller to take over a Z-wave network,'' in *Proc. Black Hat Eur.*, 2017, pp. 1–9.

[10] 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, Mar. 2016.

[11] C. W. Badenhop, S. R. Graham, B. W. Ramsey, B. E. Mullins, and L. O. Mailloux, ''The Z-wave routing protocol and its security implications,'' *Comput. Secur.*, vol. 68, pp. 112–129, Jul. 2017, doi: 10.1016/j.cose.2017.04.004.

[12] J. Hall and B. Ramsey, ''Breaking bulbs briskly by bogus broadcasts,'' ShmooCon, Washington, DC, USA, 2016.

[13] J. D. Fuller, B. W. Ramsey, M. J. Rice, and J. M. Pecarina, ''Misuse-based detection of Z-wave network attacks,'' *Comput. Secur.*, vol. 64, pp. 44–58, Jan. 2017, doi: 10.1016/j.cose.2016.10.003.

[14] K. Kim, K. Cho, J. Lim, Y. H. 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 Mobile Comput.*, vol. 66, Jul. 2020, Art. no. 101211.

[15] N. Boucif, F. Golchert, A. Siemer, P. Felke, and F. Gosewehr, ''Crushing the wave - new Z-wave vulnerabilities exposed,'' 2020, *arXiv:2001.08497*.

[16] C. W. Badenhop, B. W. Ramsey, B. E. Mullins, and L. O. Mailloux, ''Extraction and analysis of non-volatile memory of the ZW0301 module, a Z-wave transceiver,'' *Digit. Invest.*, vol. 17, pp. 14–27, Jun. 2016, doi: 10.1016/j.diin.2016.02.002.

[17] C. Badenhop, S. R. Graham, B. E. Mullins, and L. O. Mailloux, ''Looking under the Hood of Z-wave: Volatile memory introspection for the ZW0301 transceiver,'' *ACM Trans. Cyber-Phys. Syst.*, vol. 3, no. 2, p. 20, 2018. [Online]. Available: https://doi.org/10.1145/3285030.

[18] Z-Wave Alliance. *Z-Wave Products*. Accessed: Dec. 28, 2021. [Online]. Available: https://products.z-wavealliance.org/search/DoAdvanced Search?productName=&productIdentifier=&productDescription=& category=-1&brand=-1&regionId=-1&supportsS2=on&order=

[19] MITRE. *CVE-2018-19982*. Accessed: Dec. 14, 2021. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2018-19982

[20] *CVE-2018-19983*. Accessed: Dec. 14, 2021. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2018-19983

[21] M. Antonakakis, T. April, M. Bailey, and M. Bernhard, ''Understanding the mirai botnet,'' in *Proc. 26th USENIX Secur. Symp.*, 2017, pp. 1093–1110.

[22] G. Kambourakis, C. Kolias, and A. Stavrou, ''The mirai botnet and the IoT zombie armies,'' in *Proc. MILCOM - IEEE Mil. Commun. Conf. (MILCOM)*, Oct. 2017, pp. 267–272.

[23] Y. Dvorkin and S. Garg, ''IoT-enabled distributed cyber-attacks on transmission and distribution grids,'' in *Proc. North Amer. Power Symp. (NAPS)*, Sep. 2017, pp. 1–6.

[24] K. Sonar and H. Upadhyay, ''A survey: DDOS attack on Internet of Things,'' *Int. J. Eng. Res. Develop.*, vol. 10, no. 11, pp. 58–63, 2014.

[25] J. Mirkovic, S. Dietrich, D. Dittrich, and P. Reiher, *Internet Denial of Service: Attack and Defense Mechanisms (Radia Perlman Computer Networking and Security)*. Upper Saddle River, NJ, USA: Prentice-Hall, 2004.

[26] M. Zalewski. *American Fuzzy Lop*. Accessed: Dec. 14, 2021. [Online]. Available: https://lcamtuf.coredump.cx/afl/

[27] Software Engineering Institute. *The CERT Division*. Accessed: Dec. 14, 2021. [Online]. Available: https://www.sei.cmu.edu/about/divisions/cert/index.cfm

[28] C. K. Nkuba. *Proof-of-Concept Attack Scenarios*. Accessed: Dec. 14, 2021. [Online]. Available: https://youtu.be/RdVWxwg3FIE

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

[30] M. Muench, J. Stijohann, F. Kargl, A. Francillon, and D. Balzarotti, ''What you corrupt is not what you crash: Challenges in fuzzing embedded devices,'' in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2018, pp. 1–15.

[31] J. Wright. *River Loop Security: Killerbee*. Accessed: Dec. 14, 2021. [Online]. Available: https://github.com/riverloopsec/killerbee

[32] C. Bernardini, A. Lahmadi, and O. Festor, ''Development of a fuzzing tool for the 6LoWPAN protocol,'' INRIA, Villers-Lès-Nancy, France, Tech. Rep. RR-7817, Nov. 2011. [Online]. Available: https://hal.inria.fr/hal-00645948/document

[33] J. Chen, W. Diao, Q. Zhao, C. Zuo, Z. Lin, X. Wang, W. C. Lau, M. Sun, R. Yang, and K. Zhang, ''IoTFuzzer: Discovering memory corruptions in IoT through app-based fuzzing,'' in *Proc. Netw. Distrib. Syst. Secur. Symp.*, San Diego, CA, USA, 2018, pp. 1–15.

[34] Beyond Security. *beSTORM Black Box Testing*. Accessed: Dec. 14, 2021. [Online]. Available: https://www.beyondsecurity.com/solutions/bestorm.html

[35] Synopsys. *Fuzzing Test Suites*. Accessed: Dec. 14, 2021. [Online]. Available: https://www.synopsys.com/software-integrity/security-testing/fuzz-testi ng/defensics.industry.iot.html

[36] Z. Shelby, K. Hartke, C. Bormann, and B. Frank, ''Constrained application protocol (CoAP), draft-ietf-core-COAP-13,'' Internet Eng. Task Force–IETF, Orlando,FL, USA, Tech. Rep. 13, 2012.

[37] U. Hunkeler, H. L. Truong, and A. Stanford-Clark, ''MQTT-S—A publish/subscribe protocol for wireless sensor networks,'' in *Proc. 3rd Int. Conf. Commun. Syst. Softw. Middleware Workshops (COMSWARE)*, Jan. 2008, pp. 791–798.

[38] Y. Zheng, A. Davanian, H. Yin, C. Song, H. Zhu, and L. Sun, ''FIRM-AFL: High-throughput greybox fuzzing of IoT firmware via augmented process emulation,'' in *Proc. 28th USENIX Secur. Symp. (USENIX Secur.)*, 2019, pp. 1099–1114.

[39] D. D. Chen, M. Egele, M. Woo, and D. Brumley, "Towards automated dynamic analysis for linux-based embedded firmware," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2016, pp. 1–16.
[40] J. Zaddach, L. Bruno, A. Francillon, and D. Balzarotti, "AVATAR: A framework to support dynamic security analysis of embedded systems' firmwares," in *Proc. NDSS*, 2014, pp. 1–16.
[41] J.-M. Picod, A. Lebrun, and J.-C. Demay, "Bringing software defined radio to the penetration testing community," in *Proc. Black Hat USA Conf.*, 2014, pp. 1–16.
[42] J. Hall, B. Ramsey, M. Rice, and T. Lacey. *EZ-Wave*. [Online]. Available: https://github.com/cureHsu/EZ-Wave
[43] J. L. Hall, "A practical wireless exploitation framework for Z-wave networks," Air Force Inst. Technol., Wright-Patterson, OH, USA, Tech. Rep. AD1054454, 2016. [Online]. Available: https://apps.dtic.mil/sti/citations/AD1054454
[44] Silicon Laboratories. *Z-Wave Global Regions*. Accessed: Dec. 14, 2021. [Online]. Available: https://www.silabs.com/wireless/z-wave/technology/global-regions.
[45] *Z-Wave Device Class Specification*. Accessed: Dec. 14, 2021. [Online]. Available: https://www.silabs.com/documents/login/miscellaneous/SDS11847-Z-Wave-Pl us-Device-Type-Specification.pdf
[46] M. Klein. *What is Z-Wave Long Range and How Does it Differ from Z-Wave?*. Accessed: Dec. 14, 2021. [Online]. Available: https://z-wavealliance.org/what-is-z-wave-long-range-and-how-does-it-differ-from-z-wave/
[47] International Telecommunication Union. *Short Range Narrow-Band Digital Radiocommunication Transceivers-PHY and MAC Layer Specifications*. Accessed: Dec. 14, 2021. [Online]. Available: https://www.itu.int/rec/T-REC-G.9959
[48] C. Paetz, *Z-Wave Essentials*. Scotts Valley, CA, USA: CreateSpace Independent Publishing Platform, 2018.
[49] Silicon Laboratories. *Z-Wave Specification*. Accessed: Dec. 14, 2021. [Online]. Available: https://www.silabs.com/products/wireless/mesh-networking/z-wave/specification
[50] Z-Wave Alliance. *Z-Wave Transport-Encapsulation Command Class Specification*. Accessed: Dec. 14, 2021. [Online]. Available: https://www.silabs.com/documents/login/miscellaneous/SDS13783-Z-Wave-Tr ansport-Encapsulation-Command-Class-Specification.pdf
[51] Pub, NIST FIPS, "Announcing the advanced encryption standard (AES)," *Federal Inf. Process. Standards Publication*, vol. 197, pp. 1–51, 2001.
[52] R. Housley, *Using AES-CCM and AES-GCM Authenticated Encryption in the Cryptographic Message Syntax (CMS)*, document RFC 5084, Nov. 2007.
[53] D. J. Bernstein, "Curve25519: New Diffie-Hellman speed records," in *Public Key Cryptography—PKC 2006*, M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, Eds. Berlin, Germany: Springer, 2006, pp. 207–228, doi: 10.1007/11745853_14.
[54] Pen Test Partners. *Z-Shave.Exploiting Z-Wave Downgrade Attacks*. Accessed: Dec. 14, 2021. [Online]. Available: https://www.pentestpartners.com/security-blog/z-shave-exploiting-z-wave-downgrade-attacks
[55] C. Fu, Q. Zeng, and X. Du, "Hawatcher: Semantics-aware anomaly detection for appified smart Homes," in *Proc. 30th USENIX Secur. Symp. (USENIX Secur.)*, 2021, pp. 1–18.
[56] A. Makhshari and A. Mesbah, "IoT bugs and development challenges," in *Proc. IEEE/ACM 43rd Int. Conf. Softw. Eng. (ICSE)*, May 2021, pp. 460–472.
[57] Z-Wave Alliance. *Z-Wave Alliance Announces New Security Requirements for All Z-Wave Certified IoT Devices*. Accessed: Dec. 14, 2021. [Online]. Available: https://z-wavealliance.org/z-wave-alliance-announces-new-security-requirements-z-wave-certified-iot-devices
[58] K. Serebryany, "Continuous fuzzing with libFuzzer and AddressSanitizer," in *Proc. IEEE Cybersecurity Develop. (SecDev)*, Nov. 2016, p. 157.
[59] M. Ossmann. *HackRF One*. Accessed: Dec. 14, 2021. [Online]. Available: https://greatscottgadgets.com/hackrf/one/
[60] Great Scott Gadgets. *YARD Stick One-a Sub-1 GHz Wireless Test Tool Controlled by Your Computer*. Accessed: Dec. 14, 2021. [Online]. Available: https://greatscottgadgets.com/yardstickone
[61] E. Ryherd. *What is the Difference Between Z-Wave and Z-Wave Plus?* Accessed: Dec. 14, 2021. [Online]. Available: https://drzwave.blog/2018/06/13/whats-the-difference-between-z-wave-and-z-wave-plus/
[62] A. Helin. *Radamsa: A General Purpose Fuzzer*. Accessed: Dec. 14, 2021. [Online]. Available: https://gitlab.com/akihe/radamsa
[63] Wikipedia. *Fuzzing*. Accessed: Dec. 14, 2021. [Online]. Available: https://en.wikipedia.org/wiki/Fuzzing
[64] MITRE. *CWE Vulnerability Enumeration: A Community-Developed List of Common Software Security Weaknesses*. Accessed: Dec. 14, 2021. [Online]. Available: https://cwe.mitre.org/

CARLOS KAYEMBE NKUBA received the B.S. degree in computer science from the Protestant University of Lubumbashi, Lubumbashi, Democratic Republic of the Congo (DRC), in 2010, and the B.S. degree in information technology (major) and global management (minor) and the M.S. degree in computer science information technology from Handong Global University (HGU), Pohang, South Korea, in 2014 and 2016, respectively. He is currently pursuing the Ph.D. degree with the Computer Science and Engineering Department, Korea University. Prior to joining HGU, he worked as a IT Network Administrator at Gecamines Mining, DRC. His research interests include computer software and network security, fuzzing, and the IoT automated vulnerability discovery.

SEULBAE KIM received the B.S. and M.S. degrees in computer science and engineering from Korea University, in 2016 and 2018, respectively. He is currently pursuing the Ph.D. degree with the School of Computer Science, Georgia Institute of Technology. His research interests include computer systems security, automated vulnerability detection, and cyber-physical systems security.

SVEN DIETRICH (Senior Member, IEEE) is currently a Professor with the Computer Science Department, City University of New York (CUNY) Hunter College and the Graduate Center. Prior to joining CUNY Hunter College, he was with the Mathematics and Computer Science Department, CUNY John Jay College of Criminal Justice, as an Associate Professor, from 2014 to 2020. He was as an Assistant Professor with the Computer Science Department, Stevens Institute of Technology, from 2007 to 2014. He was a Senior Member of the Technical Staff at CERT and CyLab, Carnegie Mellon University, from 2001 to 2007. His research interests include computer and network security, cryptography, anonymity, and privacy.

HEEJO LEE received the B.S., M.S., and Ph.D. degrees in computer science and engineering from the POSTECH, South Korea. From 2000 to 2001, he was a Postdoctoral Researcher with the Department of Computer Sciences and CERIAS, Purdue University. He is currently a Professor with the Department of Computer Science and Engineering, Korea University, Seoul, South Korea. Before joining Korea University, he was at AhnLab, Inc., as a CTO, from 2001 to 2003. He serves as an Editor for the IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY and the *Journal of Communications and Networks*.