



**A MULTIFACETED SECURITY
EVALUATION OF Z-WAVE, A
PROPRIETARY IMPLEMENTATION OF THE
INTERNET OF THINGS**

DISSERTATION

Christopher W. Badenhop
AFIT-ENG-DS-17-J-074

**DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY**

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

DISTRIBUTION STATEMENT A
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

The views expressed in this document are those of the author and do not reflect the official policy or position of the United States Air Force, the United States Department of Defense or the United States Government. This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

AFIT-ENG-DS-17-J-074

A MULTIFACETED SECURITY EVALUATION OF Z-WAVE, A PROPRIETARY
IMPLEMENTATION OF THE INTERNET OF THINGS

DISSERTATION

Presented to the Faculty
Graduate School of Engineering and Management
Air Force Institute of Technology
Air University
Air Education and Training Command
in Partial Fulfillment of the Requirements for the
Degree of Doctor of Philosophy

Christopher W. Badenhop, B.S., M.S. (2006), M.S. (2012)

June 2017

DISTRIBUTION STATEMENT A
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AFIT-ENG-DS-17-J-074

A MULTIFACETED SECURITY EVALUATION OF Z-WAVE, A PROPRIETARY
IMPLEMENTATION OF THE INTERNET OF THINGS
DISSERTATION

Christopher W. Badenhop, B.S., M.S. (2006), M.S. (2012)

Committee Membership:

S. R. Graham, Ph.D.
Chairman

B. E. Mullins, Ph.D., P.E.
Member

Major L. O. Mailloux, Ph.D.
Member

Abstract

This work is a case-study in the security of Z-Wave, a proprietary Internet of Things (IoT) wireless substrate, integrating sensors and actuators to provide home and office automation services. While the services minimize user burden in managing applications such as security monitoring and smart-energy, they introduce a cyber-physical attack surface into the deployed environment. Because Z-Wave is proprietary, the typical consumer is unable to ascertain the security risks in installing Z-Wave devices. To increase consumer awareness, a multifaceted security evaluation is performed on the Z-Wave transceiver system on chip (SoC). While Z-Wave devices originate from many vendors, a common transceiver facilitates interconnectivity. Herein, the transceiver is assessed as an embedded system and a communication protocol stack. Prior to a security assessment, the protocol, firmware, and non-volatile memory are partially reverse engineered to lift the veil of "security by obscurity", revealing several security concerns. One example is a key extraction attack, wherein network security is compromised by extracting cryptography keys from devices lacking physical security. In another example, several discovered network protocol vulnerabilities are combined to demonstrate a Black Hole attack, where routed Z-Wave commands are silently dropped.

Table of Contents

	Page
Abstract	iv
List of Figures	ix
List of Tables	xvi
I. Introduction	1
1.1 Z-Wave, an Internet of Things (IoT) substrate implementation	5
1.2 Contributions	8
II. Passive Reconnaissance Techniques for Z-Wave	11
2.1 Introduction	11
2.2 Background	12
Z-Wave Network and Device Identifiers	12
The Z-Wave Protocol Stack	13
Related Work	17
2.3 Methodology	18
A Z-Wave Network Frame Sniffer	19
Devices Under Study	20
Passive Reconnaissance	20
2.4 Passive Reconnaissance Techniques to Discriminate between Z-Wave Devices	21
Controller and Device Pairing Information Leakage	21
Observable Controller-Specific Behavior	23
Device-Specific Acknowledgment Delay Times	25
2.5 Conclusion	32
III. Extraction and Analysis of Non-Volatile Memory of the ZW0301 Module, a Z-Wave Transceiver System-On-Chip (SoC)	34
3.1 Introduction	34
Z-Wave	35
IoT Vulnerabilities	37
Related Work	39
Contributions	40
3.2 System Under Analysis: ZW0301	41
ZM3102 Module	41
ZW0301 Architecture	41
3.3 Z-Wave Firmware Modification Attacks	45

	Page
Implications of Remote Programming	47
3.4 Methodology	48
Flash Memory Extraction	48
External EEPROM Extraction	50
Flash Memory Analysis	52
External EEPROM Analysis	53
3.5 Results of Flash Memory Analysis	54
C Format Strings as Constant Data	55
Firmware built with Keil C51 Compiler	56
INIT_MX.A51 Variable Initialization Macro	57
Optional Memory Data Blocks	58
Available Free Memory	59
3.6 Results of External EEPROM Analysis	60
Protocol Information Table	61
Node Adjacency Table	62
Controller Information Block	64
Aeon Labs Z-Stick2 Event Table	64
3.7 Conclusion	66
Future Work	67
IV. The Z-Wave Security Layer and its Susceptibility to Key Extraction and Replay Attack	69
4.1 Introduction	69
4.2 Methodology	71
Firmware Analysis	71
Protocol Analysis	73
Analysis Process	75
4.3 The Z-Wave Security Layer	75
The Secure Frame Transaction	76
Key Management	83
4.4 Attack Vectors	90
Operational Key Set Extraction	91
Replay Attack	96
4.5 Conclusion	103
V. An Analytical Black Hole Attack Model using a Stochastic Topology Approximation Technique for Reactive Ad-Hoc Routing Protocols	106
5.1 Introduction	106
Ad-Hoc Network Routing Background	110
Ad-Hoc On-demand Distance Vector Routing	111
Dynamic Source Routing	112

	Page
Related Work	113
5.2 Revisiting the Hypercube Black Hole Attack Model	115
Validation of the Attack Condition	115
Decoupling the Hypercube Topology from the Model	118
Validation of the Generalized Attack Model	120
5.3 The Attack Model Adapted for Ad-hoc Networks	122
Describing Ad-hoc Topologies Stochastically	123
Network Partitioning	124
Stochastic Analytical Black Hole Attack Model for Ad-Hoc Networks	125
Validation of Revised Model	126
Results and Analysis	128
5.4 Conclusion	129
VI. The Z-Wave Routing Protocol and its Security Implications	131
6.1 Introduction	131
6.2 Z-Wave Background	132
Pairing Operation	133
Node Identification	133
Command Class	134
Related Work	134
6.3 Reversing Engineering and Security Assessment	135
Methodology	135
Network Under Study	136
The Software Defined Radio (SDR) Test Platform	138
Collection and Injection Techniques	142
Reverse Engineering Process	143
Security Analysis Process	144
6.4 The Reverse Engineered Z-Wave Routing Protocol	145
Source Routing	145
Network Management	151
6.5 Security Analysis on Z-Wave Routing Protocol	159
Existing Security in Z-Wave	159
Vulnerabilities	162
Recommendations	174
6.6 Conclusion	175
VII. Concluding Remarks	179
7.1 Future Work	180
Volatile Memory Analysis	181
Physical (PHY) Layer Vulnerabilities	182
Trust in Evolving and Heterogeneous Z-Wave Device Configurations	183

	Page
7.2 Results of the Case Study	184
Bibliography	187

List of Figures

Figure		Page
1	The IoT device architecture.	3
2	Desired security attributes of an IoT device.	5
3	The Z-Wave IoT device architecture, highlighting regions developed by the device vendor versus regions provided by the Z-Wave Transceiver SoC and accompanying firmware.	6
4	Five Layer Z-Wave Reference Model mapped to the Five Layer TCP/IP Reference Model.	14
5	Z-Wave frame formats. Note the End Header (EHR) is only transmitted when using the 9.6 kbps data rate.	15
6	Wireshark Z-Wave dissector	19
7	The Z-Wave frame sniffer embedded within a Z-Wave network.	21
8	The network inclusion process of a Jasco smart dimmer with an Aeon Labs Z-Stick controller.	22
9	Frame capture of an Aeon Z-Stick SET command.	24
10	Frame capture of a MiCasa Verde VeraLite SET command.	24
11	The 99% confidence interval (CI) of the Acknowledge (ACK) response delay for devices from different vendors.	28
12	The 99% CI of the ACK response for devices from the same vendor.	29
13	The 99% CI of the ACK response delay at a 60 second polling interval.	31
14	ZM3102 module with ZW0301 Z-Wave transceiver, RF front-end circuit, and RF filter.	42
15	GoodFET 42 board used for ZW0301 reprogramming.	45

Figure		Page
16	Disassembly of the main function in the firmware of an Ecolink PIR sensor.	45
17	Disassembly of the modified main function in an Ecolink PIR sensor firmware image.	46
18	Verification that the firmware of the Ecolink PIR sensor is modified when compared to the factory firmware image.	47
19	GoodFET to ZM3102 interface block diagram.	49
20	Sample output of the goodfet.zensys info and dump commands.	49
21	USB controller to ZM3102 interface block diagram.	50
22	MiniPRO EEPROM programmer with SOIC8 to DIP8 converter reading a Micron 25LC256 EEPROM.	51
23	MiniPRO software displaying the contents of a Micron 25LC256 EEPROM.	52
24	Z-Wave devices with successful Flash memory exfiltration - A) Ecolink PIR sensor, B) Linear garage door controller, C) Honeywell LynxTouch module, and D) Trane thermostat.	53
25	Controllers used in study - A) Aeon Labs Z-Stick2, B) Vera-E controller, and C) Honeywell LynxTouch module.	55
26	ZW0301 firmware image layout.	56
27	Printf format strings in Trane ZW0301 Flash memory.	57
28	Program readINITSEC decoding the C_INITSEC of the Ecolink PIR sensor.	57
29	Three 128-byte data blocks found in an Ecolink PIR sensor firmware image.	59
30	Example boundary string.	61
31	Example protocol information table.	61
32	Node Adjacency Table from an Aeon Z-Stick2 EEPROM.	63

Figure		Page
33	Controller information within a Honeywell LynxTouch module.	64
34	Portion of a Z-Stick2 event table.	65
35	Event for FortrezZ Water Valve paired to a Z-Stick2.	66
36	Devices used for the study are the Yale door lock (lower-left), Aeon Multisensor4 (upper-left), and MiCasa Verde-Edge controller (right).	72
37	Beginning of the American Encryption Standard (AES) Substitution Box (S-BOX) table embedded in a Yale door lock firmware.	73
38	Beginning of the AES inverse S-BOX table embedded in a Yale door lock firmware.	73
39	Two secure frame transactions between a door lock and controller are captured with a SDR.	78
40	The nonce generating algorithm found in a Yale door lock firmware image.	79
41	The application layer fields of a secure frame in the context of a Z-Wave Physical layer Protocol Data Unit (PPDU).	80
42	The Cipher Message Authentication Code (CMAC) derivation process.	81
43	The bootstrap keys are a set of keys, common to all Z-Wave devices, used to send the controller's network key to a device during network inclusion.	84
44	Packet capture of a Vera Edge Controller providing the network key to a Yale door lock with a node ID of 25 using a secure frame transaction.	85
45	Frame 14691 is decrypted using decryptPCAPNG and the bootstrap key set to reveal a security set key message containing the network key. The network key begins at the fourth byte of the decrypted payload.	85

Figure	Page
46	Binary dump of the operational key set file found in MiCasa Verde Edge controller, shown in the context of its filesystem directory. 86
47	Segment of the Electronically Erasable/Programmable Read-Only Memory (EEPROM) of a Yale door lock containing the network key at address 0x012D. 87
48	Reversing the key derivation process to derive the Electronic Code Book (ECB) inblocks required to generate the encryption and authentication keys from the network key. 88
49	Intel 8051 disassembly of the key derivation algorithm, located in the firmware of a Yale door lock. 89
50	Internal hardware components for the Aeon Multisensor 4, where the EEPROM used by the ZW0501 is highlighted. 94
51	The network key is located at address 0x60A0 of an Aeon Multisensor 4 EEPROM. 94
52	Frames captured while OpenBarley opens a Yale door lock using a key extracted from another Z-Wave device. 96
53	Decrypted payloads of the secure frames observed as OpenBarley uses the extracted keys to successfully send a secure frame with an unlock door command. 97
54	EEPROM of an Aeon Multisensor after being excluded from the network, where the network key is zeroed from memory. 98
55	Frames captured during a replay attack attempt, where requests for the door lock state are ignored. 99
56	Decryption of the frames captured during a replay attack attempt, where requests for the door lock state are ignored. 100
57	99% CIs estimating the true mean number of rounds until collision plotted against the number of available states for a given nonce length in bits. Nonce lengths evaluated at 8, 16, 24, and 32-bit. 101

Figure	Page
58	Normalized packet loss as a function of hop-distance.....117
59	4x4 Wireless grid topology with labeled source node classes 121
60	Normalized packet loss due to Black Hole attack for the 4x4 grid network 122
61	Simulated packet loss of ad-hoc networks vs. hypercube and stochastic analytical model predictions.....127
62	Devices used in the network under study. From top to bottom, this includes an Aeon USB ZStick2, a GE Outdoor Switch, and three Aeon Appliance Switches. 137
63	The topology of the network under study used for demonstrating the results of the reverse engineering and security analysis findings. 138
64	The SDR Test Platform: The Ettus B210 SDR is shown in the lower left. The laptop has two YARD Stick One SDRs, shown on the left and right side of the laptop. 139
65	Protocol stack for SDR-based Z-Wave R2 transceiver. 140
66	Protocol stack for SDR-based Z-Wave R1 transceiver. 142
67	The Z-Wave Network Header format. 145
68	The frames observed in the network under study when a Source Route (SR) is routed from Node 1 to 3 and the corresponding route ACK taking the reverse route back to Node 1. 149
69	A protocol activity diagram of a SR routed from Node 1 to 3 and the route ACK taking the reverse route back to Node 1. 149
70	Captured frames as the result of the SDR platform injecting a SR frame into the network under study using imaginary source and destination IDs. Because Node 10 does not exist, Node 5 fails to forward the frame and generates a route No-Acknowledge (NACK) that takes the reverse route back to the source. The NACK frame indicates the fourth hop failed. 151

Figure		Page
71	A protocol activity diagram of a SR routing from Node 11 to 10. Because Node 10 does not exist, Node 5 is unable to successfully forward the frame. It generates a route NACK, which takes the reverse route back to source Node 11.	151
72	The Z-Wave Network Management Architecture.	152
73	Neighbor list primitive containing the first 16 node IDs.	154
74	SR cache entry primitive.	154
75	SR cache assignments for Node 3.	158
76	Route selections for Node 3 (Aeon Appliance Switch).	158
77	Route selections for Node 2 (GE Outdoor Switch).	159
78	SDR impersonates Node 2 by requesting the Neighbor List (NL) of Node 2, to which Node 2 replies.	163
79	SDR is adding fake Node 10 to the NL of Node 3.	164
80	Results of the SDR test platform performing the Topology Discovery algorithm on the network under study.	167
81	The routing behavior of Node 5 is modified by the SDR test platform to take an alternate route.	168
82	The SDR test platform is forwarding routed frames without error.	169
83	A route NACK is triggered when the SDR test platform forwards a frame with an invalid checksum.	170
84	Man-In-The-Middle (MITM) Attack: Changing the meaning of a forwarded frame without detection.	170
85	Black Hole attack on Z-Wave network where Node 10 is dropping frames sent from Node 5 to Node 1.	173
86	A summary of the results of security analysis on the components of Z-Wave devices with respect to assumed security attributes.	180

Figure		Page
87	Z-Wave device components covered in this dissertation.	181

List of Tables

Table		Page
1	List of salient features discovered in Flash memory.....	54
2	Data and artifacts in EEPROM of three controllers.	60
3	Nonce request rates for a Yale door lock and MiCasa Verde Edge controller.....	102
4	Values for $q_k(x)$ for each source node class in the 4x4 grid	120
5	Four simulated ad-hoc network types under test	126
6	Forwarding Targets for a SR of length N , $1 \leq N \leq 4$	146
7	Known Types of Routing Frames.....	147
8	Known network commands of the System command class.....	153

A MULTIFACETED SECURITY EVALUATION OF Z-WAVE, A PROPRIETARY IMPLEMENTATION OF THE INTERNET OF THINGS

I. Introduction

The Internet of Things (IoT) is a *global infrastructure* of interconnected, autonomous, and heterogeneous cyber-physical computing elements, coordinating to provide application services in the physical and informational domains [ITU12b]. In crudest terms, the IoT is an infrastructure that allows *anything* to communicate *anywhere* at *anytime* [ITU12b]. It shares similar properties of *ubiquitous computing*, where both offer highly available computing elements distributed over an operational region. With considerable overlap, the two topics differ in their emphasis. While IoT focuses on the properties of device elements, ubiquitous computing is most interested in system-wide properties emerging from coordinating and distributed computing elements [Oxy17]. However, when several IoT devices are composed to create a distributed application, it is difficult to label the system as exclusively IoT or ubiquitous.

Like ubiquitous computing and even cloud computing, IoT is an emerging abstraction architecture, where the user experience is well-defined and the technical challenges are addressed by the device designers. Consequently, IoT system implementations have large variation with respect to applications, interconnections, autonomous capacity, and resource requirements. For example, an IoT refrigerator hosts an embedded Linux computing element providing monitoring and reporting of sensor information, including temperature and humidity readings, ice levels, and a weight scale for the milk shelf [CR16]. Beyond sensing and reporting, the device is also able to control temperature levels and the ice dispenser motor. The computing element

possesses interfaces to communicate over the Internet, allowing it to order milk when the milk shelf weight drops below a given threshold. This fridge is considered an IoT device because it senses, actuates, and communicates with remote servers over the Internet.

A completely different IoT device implementation is an automatic pet feeder. In a Do It Yourself (DIY) project at [Ins17], instructions are provided to construct an IoT pet feeder. The device is composed of a food hopper, dispensing tube, valve motors, and an Internet-connected Raspberry Pi computing element. The computer controls the valve using the motors based on a user-defined schedule. The user is also able to manually force a feeding event using a button on the device chassis. What constitutes it as an IoT device is the user may remotely monitor feeding events and initiate feeding events over the Internet.

While the pet feeder and refrigerator service different applications, they are both examples of home automation, which is one of the high-level applications provided by IoT systems. IoT devices deployed in domestic and office areas provide automation services to minimize the user burden [ITU12b]. Legacy examples of home automation systems include climate control and security monitoring systems; however, some implementations are closed systems. Lacking interconnectivity to the global infrastructure, it remains unclear if these systems fall within the domain of IoT. With these considerations, a generalized architecture of an IoT device is realized in Figure 1. An IoT device exists within three domains: transducer, computational, and communications.

The *thing* of an IoT device is fulfilled by the components in the transducer domain, where the device senses or actuates in the local physical space. The IoT components within the transducer domain include external, proprioception, and tamper sensors. Other components include internal timers, motors, lights, and the user interface.

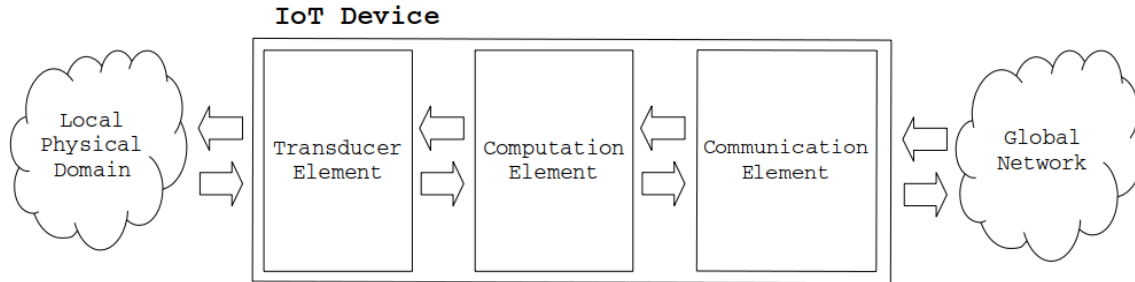


Figure 1. The IoT device architecture.

For example, a transducer component for the IoT refrigerator described above is the temperature sensor in the freezer. The motor used to control the dispensing of food pellets for the pet feeder is another example.

The components of the communication domain provide the *Internet* connectedness in IoT through a radio or wired network stack. The mechanism allows remote access to the transducer elements, such as the remote feeding capability of the pet feeder. The communication components of an IoT may also provide an interface to remotely control device configuration, request derived products from the transducer elements, and provide multi-hop routing of messages from other devices. The communication domain elements of the refrigerator and pet feeder examples are a Wi-Fi radio and IP stack.

The computational element of an IoT device contains hardware and software necessary to manage the communication and transducer elements of the device. Following the trend of modern embedded computational elements, the IoT device contains a general purpose processor, working memory, and non-volatile memory for code and configuration data. The computational element may store sensor, actuator state, topology, and configuration data, which may be consulted to perform complex, autonomous behaviors. In addition to autonomous behaviors, the processing element acts as a gateway for remote access into the local transducer components. For example, the pet feeder may have rules to feed the pet at fixed intervals. If the feeder recently

dispensed foot pellets, the device must determine if a remote request to feed the pet again is honored based on policy or is ignored because cached historical data indicates the pet is already satiated.

Security and privacy are high-level requirements for IoT devices in [ITU12b]. With respect to the IoT device architecture, desirable security attributes for each component are summarized in Figure 2. With respect to the transducer elements, the computing element must have confidence in correctness of reported sensor data and that actuating commands are correctly executed. The hardware, software, and data residing within the device may require confidentiality guarantees. For example, privacy data, intellectual property, and other secrets may reside in the memory of the IoT device, so unwanted retrieval should be prevented. Furthermore, unauthorized modification of data and code should also be prevented to maintain integrity and safety guarantees.

With respect to the communication elements of the IoT device, there are different security concerns regarding incoming and outgoing messages. Incoming messages are commands or requests for information. The device must be able to determine if the message originator has the authorization to issue the message. This capability requires source and data integrity guarantees, where the device is able to authenticate the claimed originator and detect any modification of the message in transit. Outgoing messages have slightly different security requirements. The integrity of the message must be preserved to maintain the intent of the message sent by the IoT device. Moreover, the originator needs reliability assurances that its message arrives at the destination. In some cases, the originator may desire data elements of a message to remain confidential while in transit, such as during a key exchange. Holistically, one may consider the security policy of a given IoT system is to obtain and maintain the identified security attributes throughout the system life-cycle.

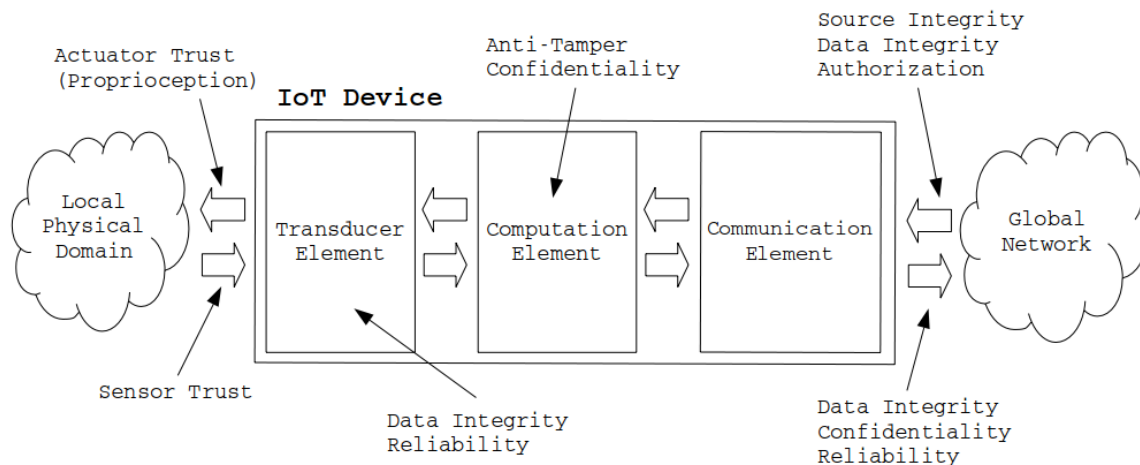


Figure 2. Desired security attributes of an IoT device.

1.1 Z-Wave, an IoT substrate implementation

When heterogeneous IoT devices are interconnected, they provide more complex services such as home automation, office security, safety monitoring, and energy conservation. Composing arbitrary IoT devices into a distributed system may incur integration challenges. Arbitrary devices differ in transducer, computational, and communication components. At worst case, the integration of N IoT devices may require a total of $N(N - 1)$ different versions of integration code for the entire system. Z-Wave is a wireless IoT substrate implementation, whose primary feature is to minimize the integration burden for the user [Z-W17]. The Z-Wave Alliance is a consortium of Z-Wave device vendors, which currently has 450 members [ZWA17]. Having a large number of vendors results in a high diversity of compatible sensing, actuating, and managing IoT devices. Z-Wave users are granted a high degree of flexibility in establishing their distributed IoT application. They may select devices from different vendors to create traditional or realize new IoT applications.

The user integration burden is reduced by requiring that Z-Wave device vendors integrate a Z-Wave transceiver System-On-Chip (SoC) into their device, which provides a common communication stack, command and control interface, and transducer in-

terface. The SoC has sufficient computing resource to facilitate many activities within the computational domain; however, it may defer more complex activities to a host processing element. Figure 3 shows the architecture of a Z-Wave device, organized into transducer, computing, and communication domains. The Z-Wave firmware, transceiver radio, and communication stack are provided to Z-Wave device vendors in the form of a Z-Wave transceiver SoC. Access to the technical details of the hardware and its associating software are restricted by the producers of the chip (i.e., it is proprietary). As of 2009, Sigma Designs is the owner of the Z-Wave SoC package after acquiring Zensys [Hig17]. Vendors gain access to the SoC by signing a Non-Disclosure Agreement (NDA) and providing a financial investment [Sig15]. The Z-Wave SoC provides the communication and computation resources, while the vendor is responsible for integrating the SoC with their transducer elements and additional computation components.

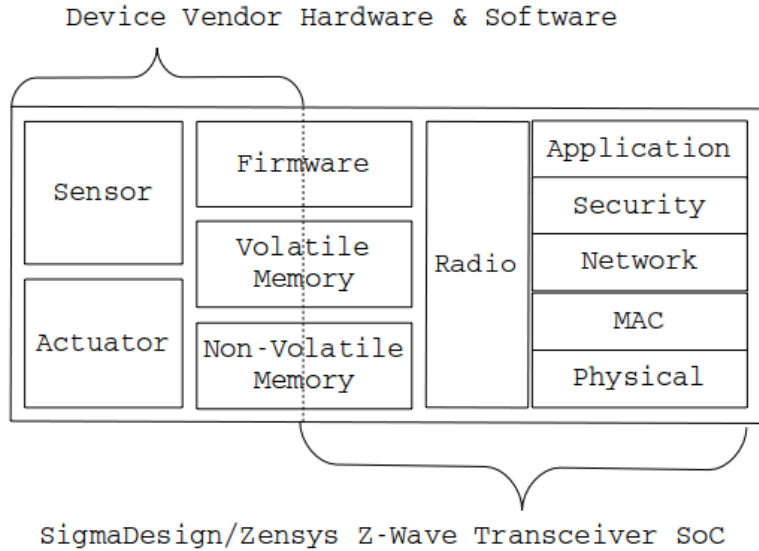


Figure 3. The Z-Wave IoT device architecture, highlighting regions developed by the device vendor versus regions provided by the Z-Wave Transceiver SoC and accompanying firmware.

While originally closed-source, the technical details on several proprietary compo-

nents have been released into the public domain. ITU-T G.9959 is the specification of the Physical (PHY) and Medium Access Control (MAC) layers of the Z-Wave protocol stack [ITU12a]. An open-source project named OpenZwave allows personal computers to act as Z-Wave controllers to manage distributed applications [Ope15b]. The source code reveals many aspects of the application layer (i.e., transducer interface) of the protocol stack.

Since security is one of the high-level requirements for IoT devices in [ITU12b], the existence of the security attributes provided by Z-Wave technology is an important, yet unanswered question. While proprietary restrictions offer marginal security through obscurity, they create an obstruction towards the public capacity to evaluate the security state of Z-Wave. Consequently, the public is forced to trust that Z-Wave devices installed in their homes and offices correctly implement security and privacy. Blind trust of closed-systems continues to prove to be a fallacious endeavor [KYSL14, McK12, YB11, HHS⁺06, HMT06, Cre15, Sch14, Pal13, JG08]. An alternative approach to blind trust is to lift the veil of the proprietary restrictions through reverse engineering. Once significant details of the system are revealed, a candid security assessment can be performed.

There is no shortage of IoT abstract security architectures, challenges, and solutions [MYAZ15, GS16, EKA15, ACH15]; however, few have applied a security analysis to an IoT implementation. Of the few exceptions, a security analysis is performed on an RFID-based IoT system [KZA⁺16], and an IoT system tracking ambulance dispatching and patient status is evaluated in [STM16]. To date, a comprehensive security evaluation of Z-Wave has not been performed. Not only will such an activity benefit Z-Wave, it provides a case for applying a practical bottom-up approach towards IoT security to compliment the well-researched top-down approach.

1.2 Contributions

The contributions of this work are as follows. A security assessment is performed on several components of the Z-Wave device architecture to determine the conditions where the system fails to maintain the desired security attributes highlighted in Figure 2. The work herein is not an exhaustive security assessment of Z-Wave; rather, it is motivated by considering the most common proprietary components. The common proprietary components of specific interest include the firmware, non-volatile memory, and communication protocol stack provided by the Z-Wave Transceiver SoC. While the transducer elements are proprietary, the scope of a discovered security violation is device specific; therefore, a security analysis of transducer elements are ignored for this effort. Prior to a security assessment, the proprietary components, including the firmware, device memory, and protocol stack are reverse engineered using white and black box analysis techniques [SGA07] such as firmware static analysis and protocol fuzzing. The results of this analysis constitutes a detailed, engineering focused case study of IoT security. The contributions in each chapter are highlighted below.

In Chapter II, the application, MAC, and PHY layer of the communication stack are evaluated for their susceptibility to passive reconnaissance for Z-Wave devices used in critical infrastructure applications. Several device fingerprinting and enumerating techniques are identified to include noting differences in protocol behavior and the inter-frame time delay between a sent frame and its Acknowledge (ACK). It is also discovered that the application layer provides a capability reporting mechanism, which reveals a significant amount of information about the device. This chapter is published as part of a book chapter in Critical Infrastructure Protection IX [BFH⁺15].

In Chapter III, the firmware and non-volatile data memory of a set of devices are reverse engineered. Several data structures are identified, including the location of the device identifiers, a network adjacency table, and a list of nodes along with their

reported capabilities. The firmware compiler is identified, along with several memory segments. A firmware modification attack is demonstrated on a Z-Wave device. The chapter is published in the Elsevier Journal of Digital Investigation [BRMM16].

The Z-Wave communication stack has a security layer to provide integrity, replay, and confidentiality guarantees for application messages. In Chapter IV, the security layer is reverse engineered and evaluated, building on the work performed in [FG13]. The key derivation algorithm, nonce generating algorithm, and memory locations of the security keys are discovered in Z-Wave devices and reverse engineered using static analysis. A key extraction vulnerability is exposed, where physically unsecured devices may be used to learn network security keys to send authenticated and encrypted commands to physically secured devices. The replay prevention mechanism is also evaluated, where it is found to be resistant to a brute-force nonce collision attack. With the exception of the replay attack reverse engineering and analysis, the chapter is published in the hacker trade journal POC ||GTFO [BR16].

Chapter V details work in ad-hoc network protocol security, where an analytical model is able to predict the effects of a Black Hole Attack on several simulated network topologies before they are deployed. The chapter is published in the International Journal of Network Security [BRM16]. The knowledge gained in this chapter is applied to Chapter VI, where the topology management and forwarding mechanisms of the network layer of the communication stack are reverse engineered. A security assessment of network layer reveals additional security vulnerabilities. An outsider is able to conduct integrity-based attacks on topology state and routed frames of a Z-Wave network. The vulnerabilities are exploited to demonstrate a Black Hole attack on a Z-Wave real-world network, which is able to covertly drop application frames. Chapter VI is published in the Elsevier Journal of Computers & Security [BGR⁺17].

Chapter VII provides closing remarks for the work herein. The contributions are summarized with respect to Figure 3. Several lessons-learned from the case study and future work, including the formulation of a fourth journal paper, are discussed.

II. Passive Reconnaissance Techniques for Z-Wave

2.1 Introduction

Z-Wave provides IoT services within the home and office automation domains. The proprietary technology is marketed and maintained by the Z-Wave Alliance, consisting of 450 different device vendors [ZWA17]. Z-Wave product development is restricted by nondisclosure and confidentiality agreements, stifling open-source security and resiliency research. As a result, the security implications of Z-Wave networks in home, office, and industrial applications are not yet well understood. The approach herein is to commence the security assessment of Z-Wave by analyzing the communication protocol’s susceptibility to reconnaissance activity. Reconnaissance lays the foundation for more sophisticated follow-on attacks and security analysis [SL02].

A reconnaissance activity occurs when an attacker gains information about a network through extrinsic observation. The information received can include protocols in use, types of devices on the network, traffic flow patterns, and encryption keys. Information received via reconnaissance may prove useful in providing the attacker with an accurate mapping of the system, services, or vulnerabilities enabling future, more significant malicious activity. Network reconnaissance is accomplished by capturing and analyzing wireless network traffic (a.k.a, sniffing). Using a directional, high-gain antenna, RF observations can be achieved from long distances, allowing the attacker to remain inconspicuous while gathering information. A proven exploitation of a wireless sensor network used in critical infrastructure is presented in [AH13], where their exploitation is demonstrated against three devices from different manufacturers at a maximum distance of 40 miles.

The contributions of this work are twofold. First, a Software Defined Radio (SDR)-based passive Z-Wave frame collector is developed to observe and parse Z-Wave net-

work traffic. Second, the sniffer is used to evaluate the discriminatory information leaked, without external invocation, by Z-Wave devices. Passive reconnaissance is of particular focus because it is more stealthy than active reconnaissance; it does not reveal the presence of an outsider. Moreover, initially studying passive techniques helps identify the intelligence gaps where active probing is necessary. The remainder of this chapter is as follows. Section 2.2 provides an overview of the Z-Wave communication protocol stack. The section also identifies the existing Z-Wave sniffers available to the public and other related work. Section 2.3 presents the Z-Wave sniffer configuration and devices used for the work herein. Section 2.4 elucidates the discovered discriminatory information resulting from passive reconnaissance of a Z-Wave network. The chapter conclusion and suggested future work is covered in Section 2.5.

2.2 Background

In this section, public domain knowledge on Z-Wave is summarized. This includes device identifiers, network identifiers, and the communication protocol stack [ITU12a, Pae13, Ope15b]. Prior work on Z-Wave security and frame sniffing is also included in the section.

Z-Wave Network and Device Identifiers.

Each Z-Wave network is identified by a unique 32-bit `Home ID` which is preprogrammed by the manufacturer on each control node. The `Home ID` allows multiple networks to operate within close proximity without overlap. While a network may contain multiple control nodes, only one may be designated as the primary controller and its `Home ID` is used to uniquely identify the network. During the network inclusion process, the primary control node assigns the new node an 8-bit `Node ID` which is only unique within the scope of local network.

Each device is defined by three device classes, which are the basic, generic and special device classes [Pae13]. The basic device class defines the type of device, whether controller, slave, or routing-slave. The generic device class defines what function the device is supposed to perform as a controller or slave. Lastly, the special device class allows for more specificity in the device functionality.

The Z-Wave Protocol Stack.

The Z-Wave protocol consists of five layers as shown in Figure 4. The ITU-T G.9959 recommendation specifies the PHY and MAC layers for short-range, narrow-band digital radio communication transceivers [ITU12a]. The PHY layer controls access to radio frequency medium and the MAC layer handles transmission and reception of frames between adjacent nodes. The network layer provides a multi-hop routing capability throughout the mesh. The security layer allows designated application payloads to be encrypted and signed, providing confidentiality and data integrity guarantees. The application layer contains the commands to be executed by the end device. When compared to the IP stack, Z-Wave is missing a transport layer service to handle flow control and out-of-order reassembly of fragmented frames. Since Z-Wave application layer transactions are typically one or two messages and do not exceed the maximum frame size limit, a transport layer may not be necessary. While the IP stack does not have an explicit security layer, IPsec provides a similar capability [LMM⁺12].

PHY layer.

All Z-Wave networks operate in unlicensed frequency bands: 908.4 MHz in North America, 860.4 MHz in Europe and additional frequencies in other regions. Data rates of 9.6 kbps (Rate 1), 40 kbps (Rate 2) and 100 kbps (Rate 3) are supported,

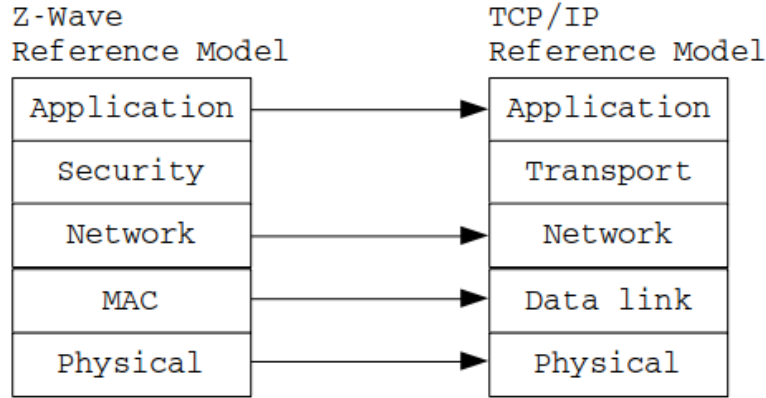


Figure 4. Five Layer Z-Wave Reference Model mapped to the Five Layer TCP/IP Reference Model.

depending on the transceiver type. The protocol offers data rates of (i) 9.6 kbps using Frequency Shift Keying (FSK) with Manchester encoding, (ii) 40 kbps using FSK with Non Return to Zero (NRZ) encoding and (iii) 100 kbps using Gaussian FSK with NRZ encoding.

As shown in Figure 5, the PHY Physical layer Protocol Data Unit (PPDU) consists of three main parts. The frame begins with a Start Header (SHR), which consists of a preamble for symbol and bit synchronization followed by a Start of Frame Delimiter (SFD). The frame payload, or Physical layer Service Data Unit (PSDU), follows. Finally, for 9.6 kbps data rate transmissions only, the frame concludes with an End Header (EHR) [ITU12a].

MAC layer.

The MAC layer uses carrier sense multiple access with collision avoidance consistent with a random back-off algorithm [ITU12a] to control access to the wireless medium. Frame collision detection is accomplished using a frame checksum and acknowledgement. A recipient only replies with an ACK when a frame is received without a detected bit error. If the sender does not observe an ACK within a given

time period, the sender may conclude the frame is not correctly received, if at all, and retransmit. While this technically detects collisions, it is actually detecting the event where a frame is transmitted over a medium with a Signal to Noise Ratio (SNR) that is conducive to bit errors. A given failed transmission cannot be causally linked with a frame collision; instead, the cause may be explained by a wide variety of factors degrading signal power or increasing noise power.

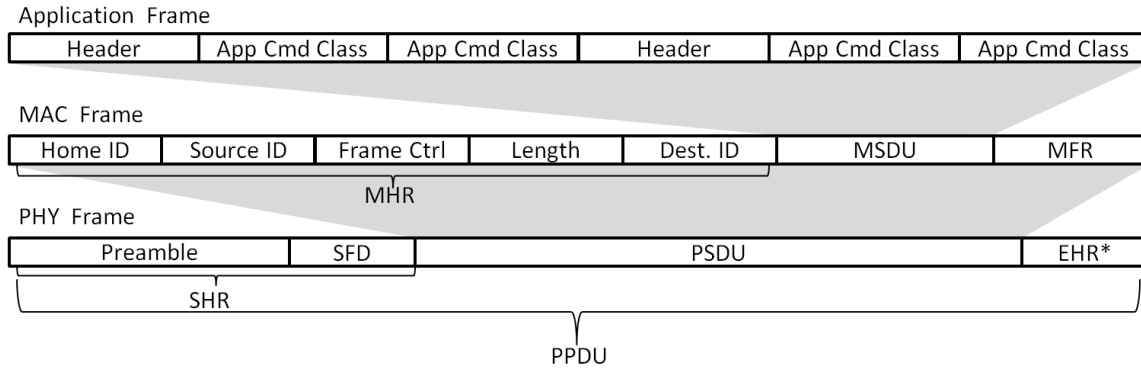


Figure 5. Z-Wave frame formats. Note the EHR is only transmitted when using the 9.6 kbps data rate.

There are three basic types of MAC frames (i.e., MAC layer Protocol Data Units (MPDUs): unicast, ACK, and multicast. Each frame type follows the same general layout shown in Figure 5 with a MAC layer Header (MHR), MAC layer Service Data Unit (MSDU), and a MAC layer Footer (MFR). Unicast frames are transmitted to only one destination address (including the broadcast address). ACK frames are structured identically to unicast frames, but they have an MSDU of zero-length and are sent in response to unicast frames (not including broadcast). Multicast frames are sent to multiple destination nodes without acknowledgments.

The MHR contains the **Home ID**, **Source ID**, a frame control field, frame length, and **Destination ID** (or bitmask in the case of multicast frames). Depending on the data rate used, The MFR contains either an 8-bit checksum or a 16-bit cyclic redundancy check.

Network layer.

Z-Wave mesh network topologies are managed by the network layer which ensures messages are successfully routed among control and slave nodes. The protocol specifies a maximum of 232 nodes and only one primary control node, although multiple secondary control nodes may exist in order to partition the network [Pae13]. Each node, with the exception of battery-operated devices, participates in routing by forwarding frames between control nodes and slave nodes outside of direct wireless transmission range. The protocol also specifies a maximum of five hops between the primary control node and any other node [Pae13].

The participating nodes of the network layer are responsible for maintaining a topology model of the network and realizing routes between nodes. A routing table is built by the primary control node based on information received, upon inclusion or request, from each slave node about the neighbors of each node [Pae13]. The Z-Wave protocol stack supports automatic topology discovery and healing to optimize routing tables when the location of a node has changed or it has been removed from the network.

Security layer.

The Z-Wave protocol supports encryption using the American Encryption Standard (AES) with 128-bit keys. When implemented, the application frame is encrypted and an 8-byte authentication frame header is appended to the end of the MSDU. While data encryption is supported by the protocol, its implementation is left to the manufacturer, who must decide whether the device transmission is sensitive enough to warrant encryption and authentication. In wireless sensor networks, both memory and power are scarce resources, discouraging developers from implementing encryption “unless required” [Fre08].

Application layer.

The application layer is partitioned into command classes, where each command class contains a well-defined set of application layer commands. A given device declares which command classes it supports when joining a network and is expected to properly respond to the commands of the supported command classes. An example of a command class is the *Binary Switch* Command Class. The binary switch uses three commands: **SET** to turn a device on or off, **GET** to request a device status and **REPORT** to respond to the request. These three commands are observed to be used in most command classes.

The Z-Wave application layer frame consists of the header, command class information and command parameters (Figure 5). The application layer is responsible for executing the commands passed to it.

Related Work.

Two published demonstrations of Z-Wave device exploitation have appeared to date, both of which relied heavily on the ability to conduct reconnaissance against the target network in order to gain requisite knowledge for crafting follow-on attacks. In [FG13], the authors gained detailed understanding of how a specific door lock implemented encryption and authentication. Using this information, they were able to discover an implementation flaw which is exploited. In [PLD14], the authors discovered the specific commands and associated bit values to toggle an alarm.

Several groups have developed sniffers capable of intercepting and transmitting Z-Wave frames [FG13, PLD14]. The Sigma Designs company markets a closed-source development kit for Z-Wave device developers [Sig15]. The kit comes with several hardware development platforms, technical documentation, build tools, and a Z-Wave protocol sniffer. While ideal, the kit requires compliance with a NDA, which may

discourage academic use of these tools.

A second sniffer project is known as z-force [FG13]. The sniffer includes custom firmware hosted on a CC1110 development board [TI14] and a z-force PC application. At the time of this writing, z-force has only been demonstrated on European Z-Wave frequency bands. The closed-source nature of the z-force project, possibly in compliance with NDA restrictions, makes it difficult to evaluate and extend.

A third Z-Wave sniffer is called Scapy-Radio and is part of an open-source *hackrf* project [PLD14]. Scapy-Radio integrates Scapy, a Python environment for manipulating network traffic and GNUradio, an open-source signal processing toolbox. The tool includes GNUradio-companion implementations of Z-Wave, Bluetooth, and ZigBee transceivers. The Z-Wave implementation receives samples from a software-defined radio, demodulates, performs Start of Frame (SOF) detection, and extracts the MPDU. As with z-force, this implementation is tuned to European bands, but may be easily modified to use other bands using GNUradio tools because it is open-source.

Several application layer sniffers are provided via OpenZWave [Ope15b]. OpenZWave is an open-source project that provides libraries and drivers to communicate with USB-based Z-Wave controllers, such as the Aeon Labs Z-Stick S2. OpenZWave provides a network querying tool to demonstrate their library API called *MinOZW*. The `python-openzwave` package, an open-source Python wrapper for OpenZWave, provides a Python version of *MinOZW* and a more powerful Z-Wave shell interface to interact with the Z-Wave controller.

2.3 Methodology

In this section, the Z-Wave sniffer architecture, devices under study, and collection technique are identified for the work herein.

A Z-Wave Network Frame Sniffer.

While one may set up Scapy or a netcat listener to collect the encapsulated User Datagram Protocol (UDP) frames sent over the local loopback device, one may also pass the captured frames to a packet dissector to minimize the burden of analyzing frame captures. Wireshark is capable of intercepting UDP datagrams, but is not able to decode the encapsulation header or Z-Wave MPDUs [Fou17].

To conduct research for this work, a custom packet dissector is developed within the Wireshark development environment to parse the Scapy-Radio encapsulation header and Z-Wave MHR. An example dissection of a Z-Wave frame is shown in Figure 6. In addition to the MAC header, the dissector also decodes the first byte of the application layer payload to identify the command class and validates the checksum. Dissectors for every command for all command classes, as identified in the OpenZWave source code, are a work in progress.

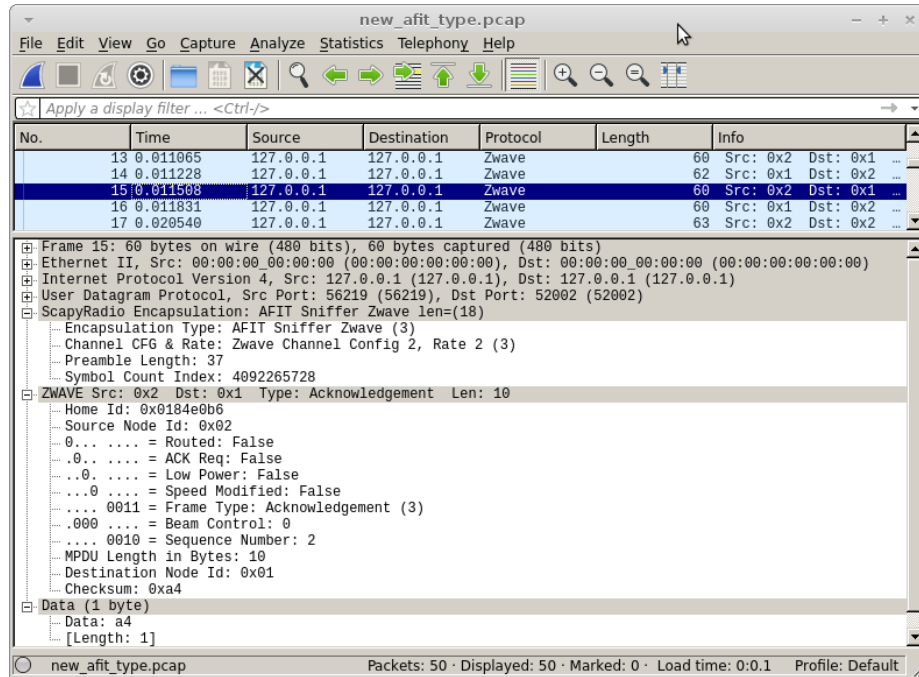


Figure 6. Wireshark Z-Wave dissector

The Z-Wave network frame sniffer architecture is a composition of a N210 Uni-

versal Software Radio Peripheral (USRP) SDR, GNUradio, Scapy-Radio, and the custom Wireshark sniffer. Wireshark is capable of exporting the captured frames into a Packet Capture (PCAP) format for off-line analysis.

Devices Under Study.

Several devices are composed in a variety of network topologies for the work herein, to include two Z-Wave controllers and four transceiver devices. The MiCasa Verde VeraLite is a stand-alone Z-Wave primary controller [Ver17]. The Aeon Labs Zstick-2 is a USB dongle enabling a PC to act as a primary controller . The Jasco Dimmer and Aeon Appliance Switch act as controllable power outlets, allowing devices plugged in to either switch to be managed over the Z-Wave network [Aeo17, Jas17]. The FortrezZ water-valve allows the valve state to be controlled over Z-Wave [For15]. Its primary application is to be used with water sensors to automatically disable a home water supply in the event a leak is detected [For15]. The Everspring light switch interfaces between a lamp socket and ordinary light-bulb, allowing the lamp to be controlled via the Z-Wave infrastructure [Eve17]. While the devices provide different services, all six devices use the ZM3102N Zensys transceiver SoC, which executes the Z-Wave protocol stack behavior.

Passive Reconnaissance.

To capture arbitrary Z-Wave frames, a receiver chain is constructed using a USRP, GNUradio, Scapy-Radio and a Z-Wave Wireshark packet dissector. A high-level diagram of the architecture embedded within a Z-Wave network is shown in Figure 7. The USRP is located within transmission range of the other Z-Wave devices. When active, the USRP collects detected frames in time ordered PCAP files. Since this is passive reconnaissance, the USRP never transmits.

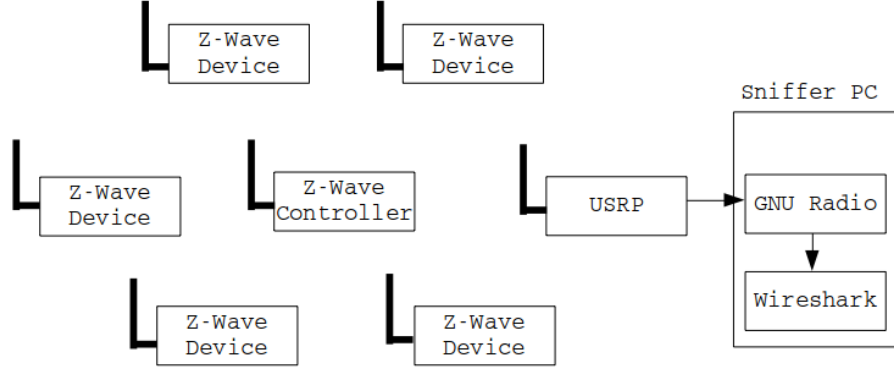


Figure 7. The Z-Wave frame sniffer embedded within a Z-Wave network.

2.4 Passive Reconnaissance Techniques to Discriminate between Z-Wave Devices

A basic invariant threat to a system, from which other threats originate, is passive reconnaissance. In this section, several low-complexity passive reconnaissance techniques are identified to further the exploration of security implications of the Z-Wave security.

Controller and Device Pairing Information Leakage.

At the end of a network inclusion event, the controller interrogates the new device to learn about it. This information is used by the controller to present accurate control options and device information to a user. Observing the responses of the joining device to the controller reveals a great deal of information about the hardware, software and current state.

Figure 8 shows the a frame capture of a Jasco dimmer switch joining a Z-Wave network managed by a a Z-Stick controller. The data in the unparsed portion of the application payloads is manually reverse engineered by referencing the OpenZWave command class source code and is summarized to the right of the Wireshark frame capture in Figure 8. The figure shows a sequence of **GET** commands from the controller

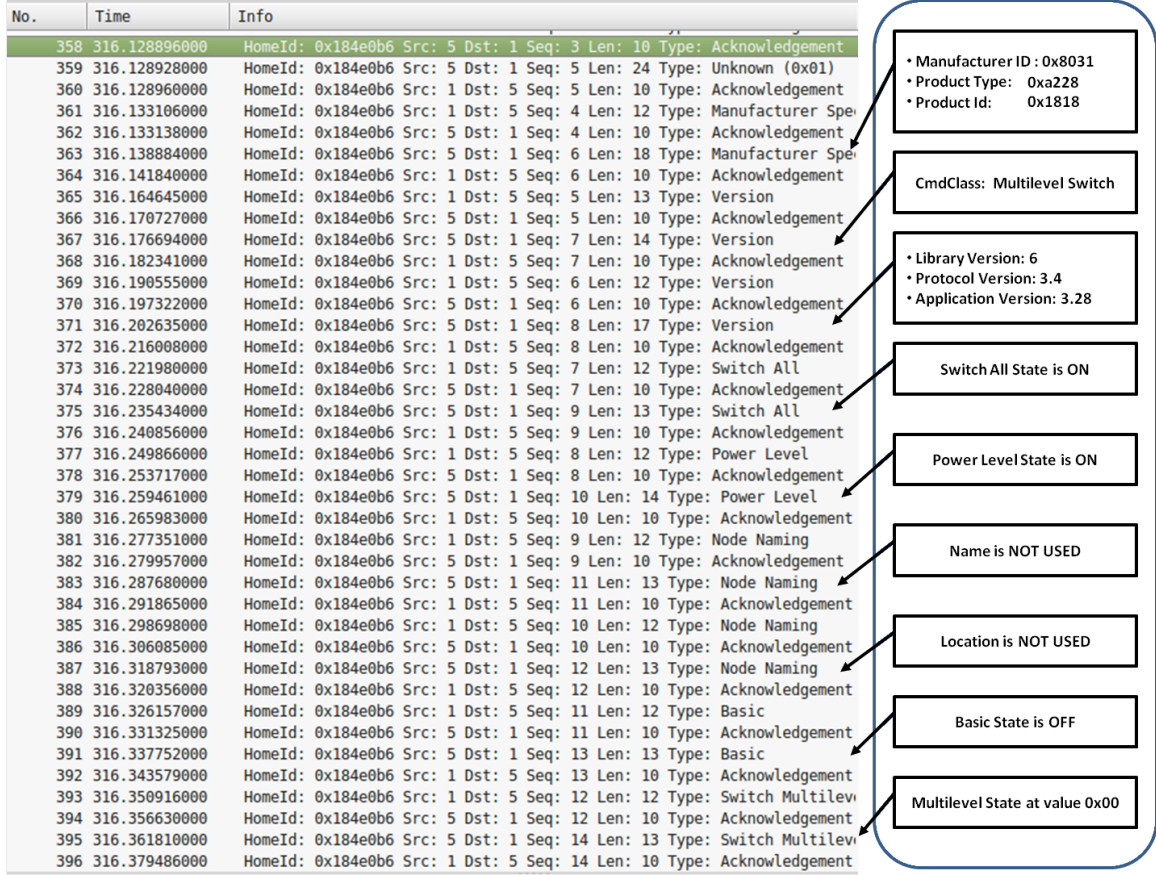


Figure 8. The network inclusion process of a Jasco smart dimmer with an Aeon Labs Z-Stick controller.

to the dimmer and, for each request, the dimmer responds with a REPORT message revealing a set of information. Device level information, such as the vendor, device type and ID are provided in a Manufacturer Specific REPORT message. The controller requests the version of the primary class of the device; in this case, the device replies with a version value of 1.0. The controller requests software version information, to which the dimmer device replies with a Z-Wave library, protocol and application version. After the device and software level information is gathered, the controller queries the device on the current state of each configuration and control item. This state information can be used by an outside observer to determine the set of messages that the device services, which is especially useful when the device specific data refers

to an unknown product type or manufacturer ID. The location state is of particular interest. While it is not initialized for the capture in Figure 8, it is possible for a device to reveal some information about its configured location. The location information is in the form of a grouping index that corresponds to a user-defined label assigned at the controller interface. A user may group devices by room, floor, or functionality so additional information is required to understand the semantics of this value. Even without knowing the physical location of a given location index, a hidden relationship is implied for all devices using it.

If the network inclusion is not observed, passive reconnaissance may still similarly capture interactions between the controller and the devices. As with the latter portion of the pairing process, the command classes of observed messages provide some information about the device’s function. The drawback is that the observation period will be significantly longer than if observed during the pairing process. This is especially true for low-power event triggered devices. These devices operate at low duty-cycles to extend battery life-time and only emit Z-Wave frames when a particular monitored event occurs, such as glass breaking, a door opening, or excessive moisture being present.

Observable Controller-Specific Behavior.

A controller may exhibit discernible differences in how it interacts with devices. To illustrate this observation, the dimmer (i.e., a multilevel switch class device) is paired with two different controllers to show that the observed frames are noticeably different.

For the first case the Jasco Smart Dimmer is paired with an Aeon Labs Z-Stick S2 controller (node 1). The multilevel switch **SET** command is executed through an OpenZWave control panel while sniffing the Z-Wave frames using Scapy-Radio’s

No.	Time	Protocol	Length	Info
52	2.344342000	Zwave	63	HomeId: 0x184e0b6 Src: 1 Dst: 5 Seq: 8 Len: 13 Type: Switch Multilevel
54	2.353639000	Zwave	60	HomeId: 0x184e0b6 Src: 5 Dst: 1 Seq: 8 Len: 10 Type: Acknowledgement
56	2.366168000	Zwave	62	HomeId: 0x184e0b6 Src: 1 Dst: 5 Seq: 9 Len: 12 Type: Switch Multilevel
58	2.368858000	Zwave	60	HomeId: 0x184e0b6 Src: 5 Dst: 1 Seq: 9 Len: 10 Type: Acknowledgement
60	2.375081000	Zwave	63	HomeId: 0x184e0b6 Src: 5 Dst: 1 Seq: 6 Len: 13 Type: Switch Multilevel
62	2.380123000	Zwave	60	HomeId: 0x184e0b6 Src: 1 Dst: 5 Seq: 6 Len: 10 Type: Acknowledgement

Figure 9. Frame capture of an Aeon Z-Stick SET command.

USRP sniffer and the custom Wireshark Z-Wave dissector. The resulting capture is shown in Figure 9. Note that the Z-Stick controller has an ID of 1 and the dimmer has a node ID of 5. The Z-Stick controller sends a multilevel dimmer command to set the dimmer to a value, shown as frame 52, which is acknowledged by the device. After receiving the acknowledgment, the controller requests an update of the dimmer’s current state (i.e., the dimmer’s current output voltage level) by sending frame 56. The dimmer complies by sending its current value to the controller in frame 60. Including the acknowledgments, this results in six frames exchanged per transaction. This same pattern is observable when the Aeon Z-Stick controller sends commands to other devices. Essentially, the controller issues a **SET** command, followed by a **GET** command on the same value to confirm that the device complied with the command. The targeted device responds to the command with a **REPORT** command.

No.	Time	Protocol	Length	Info
63	8.171197000	Zwave	63	HomeId: 0x17b784d Src: 1 Dst: 5 Seq: 3 Len: 13 Type: Switch Multilevel
65	8.272984000	Zwave	63	HomeId: 0x17b784d Src: 1 Dst: 5 Seq: 4 Len: 13 Type: Switch Multilevel
67	8.279402000	Zwave	60	HomeId: 0x17b784d Src: 5 Dst: 1 Seq: 4 Len: 10 Type: Acknowledgement

Figure 10. Frame capture of a MiCasa Verde VeraLite SET command.

For the second case, the same dimmer device is paired to a MiCasa Verde VeraLite controller. Through the VeraLite controller’s web interface, the multilevel switch **SET** command is invoked while capturing Z-Wave frames. The results of this capture are shown Figure 10. As a coincidence, the node IDs are identical to the previous case. The VeraLite controller sends two versions of the same multilevel switch **SET** command. For frame 63, the acknowledge required bit in the frame control field is not

set; however, it is set for frame 64. Once the dimmer device receives frame 64, it acknowledges its receipt to the controller, shown in the figure by the matching sequence numbers. The reason the controller initiates the same command with and without the *acknowledgment required* bit set is likely so that the controller is compatible with devices that are not capable of replying. Devices that are unable to transmit may ignore frames that require acknowledgment. As is the case with the Aeon Z-Stick controller, this pattern is observable when the VeraLite controller issues **SET** commands to devices. The pattern is that it simply issues the **SET** command with and without the acknowledgment and does not necessarily expect a reply from the device.

The results show that there is more than one way to execute a switch **SET** transaction. While it cannot be concluded that the observed patterns are unique to the controllers under study, by deduction, a given command pattern reduces the set of possible identities of the target controller. For example, an unknown controller exhibiting a pattern consistent with that of a Z-Stick controller implies that the unknown device is not a VeraLite controller.

Device-Specific Acknowledgment Delay Times.

In the event that the application layer of the Z-Wave frame is encrypted, passive reconnaissance of a device using the previously identified methods may prove difficult. By application level encryption, the command class fields are hidden, making it more difficult to identify controller specific command patterns. The outside observer must turn to non-encrypted observations to deduce properties of the devices. Implementation differences between hardware, software, or the physical environment of the devices may result in observable differences outside of the encrypted traffic. Existing techniques such as traffic analysis and side channel analysis have already been shown to be effective in thwarting confidentiality mechanisms like encryption [LH12]. Like-

wise, techniques preventing this kind of analysis are also being researched, such as those in [CMCJ04] and [YYH⁺11].

A ZigBee fingerprinting technique, which may be applicable to Z-Wave devices, exploits the implementation differences between receivers with respect to how they handle preamble sequences [RSM⁺15]. Unfortunately, it has been difficult to find Z-Wave devices using a transceiver other than the ZM3102N to verify this technique. Since the hardware is presently homogeneous, the firmware implementations are examined behaviorally to identify differences due to implementation or functionality. The ZM3102N has 32 kilo-bytes of flash memory, so each vendor has the ability to include vendor-specific behavior into their devices.

One ubiquitous observable behavior below the application layer is the measure of time for a node to send an acknowledgment upon receipt of a frame. Time of flight fingerprinting metrics are explored in [SDM12], where the authors report that variance of the measurements are due to the hardware and environmental factors. They used several PCs to perform packet injection, ACK response and time measurements. Each PC had an operating system managing independent operations that also competed for the CPU clock, resulting in variations in time of flight measurements. While this is a valid issue, Z-Wave devices are far more specialized than a PC, having application specific Input/Output (IO) and fewer, if any, concurrent tasks. This suggests that an acknowledgment response of a Z-Wave device, being an embedded device, is more deterministic than a general purpose PC. If true, then that would make ACK-based fingerprinting a valid approach to identifying Z-Wave devices.

The following experiments demonstrate that ACK-based fingerprinting of Z-Wave devices has merit. Note that while the Z-Wave devices under study are all embedded systems, the sniffer is a PC. As such, it incurs a degree of the dynamics identified in [SDM12]. The ACK delay time is measured using the packet arrival times observed by

the Wireshark Z-Wave dissector. The time values are determined by the Wireshark process when an encapsulated Z-Wave frame is observed on the *localhost* interface. Encapsulated frames are sent over the *localhost* interface by the GNUradio process. The GNUradio process receives samples from the USRP Field-Programmable Gate Array (FPGA) through a Gigabit Ethernet interface. Each process, interface and device driver contends with other resources on the sniffer PC and is subject to contention dynamics. Regardless of these dynamics, sufficient sampling of ACK delays removes these confounding effects from the measurement. In all of our experiments, exactly 70 ACK delay measurements are collected for each device under study to remove sampling bias and to facilitate comparisons of observations between the devices.

Acknowledgment Times for Different Vendor Devices.

In the first experiment, a MiCasa Verde VeraLite controller is used to repetitively issue a command to a FortrezZ water-valve. Each command generates several acknowledged frames from both the controller and the valve. Using the sniffer, sufficient frames are captured so that there are at least 70 pairs of frames with acknowledgments from both devices. The acknowledgment of a given singlecast frame is uniquely identifiable because the source field of the acknowledgment frame matches the destination field of the singlecast frame and the sequence numbers are also equal for both frames. This *uniqueness*, however, is highly transient because the frame format for data rates 9.6 kbps and 40 kbps only support 4-bit sequence numbers, so repetitive commands between the same two components result in multiple instances of pairs that appear identical but occur at different times. After carefully pairing the singlecast frames with their associative acknowledgment, the acknowledgment response time is measured as the difference in time between the observation of the singlecast frame being sent and the observation of the acknowledgment being sent. The distances between

the water-valve, VeraLite controller and the Z-Wave sniffer are all within 1 meter of each other, so propagation differences are assumed to be negligible.

Figure 11 shows the 99% confidence intervals estimating the true mean of the acknowledgment response times for the MiCasa Verde VeraLite controller and the FortrezZ water-valve. The mean response times are different with a two-sample t-test p-value of 0.00; thus, the response time of the water-valve is greater than the VeraLite controller’s response time. Moreover, the difference in the response times are an order of magnitude beyond variations due to thermal effects [Zen07]. Since both devices use the same hardware, the differences may be due to firmware implementation differences.

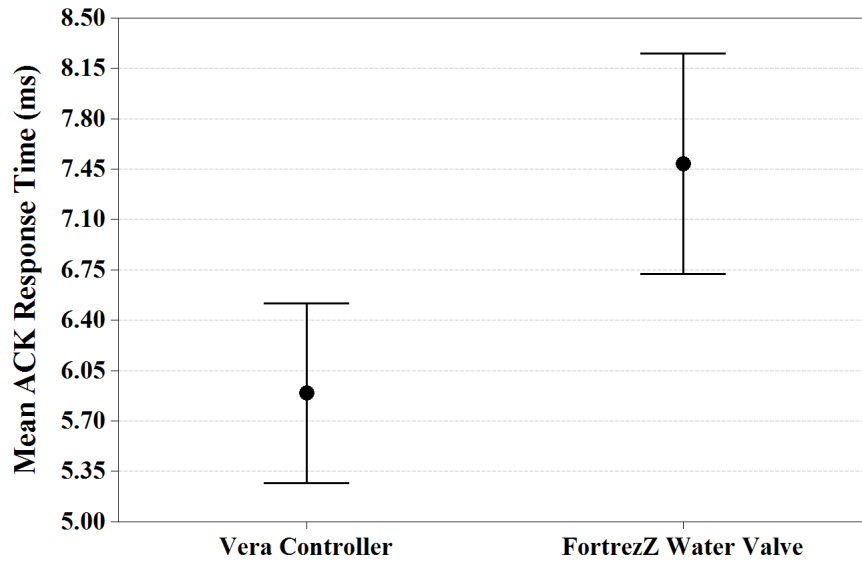


Figure 11. The 99% confidence interval (CI) of the ACK response delay for devices from different vendors.

Acknowledgment Times for Devices of the same Vendor.

The same experiment is repeated as above, but in this case the selected controller and device originate from the same vendor. The Aeon Labs Z-Stick S2 controller is used as the controller and an Aeon Labs Appliance Switch is used as the device. At

least 70 samples of acknowledgments for both systems are collected by repetitively issuing commands through the controller. Figure 12 shows the resulting 99% confidence intervals estimating the true mean acknowledgment response times for both devices. In this case, the response times of the two devices are very similar, having a two-sample t-test p-value of 0.64. This is suggestive evidence that both devices use similar firmware on the ZM3102N. The Z-Stick controller is a USB powered device that interacts with the host PC through OpenZWave, which may explain the increased variance.

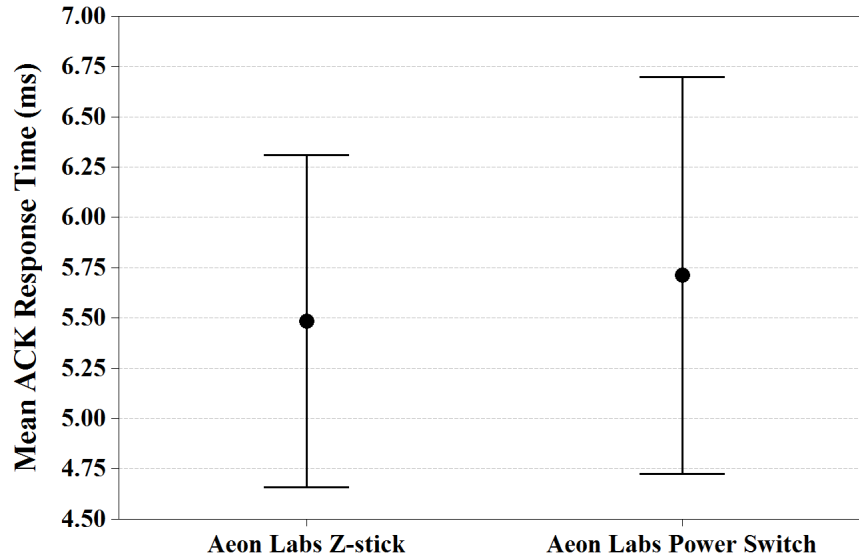


Figure 12. The 99% CI of the ACK response for devices from the same vendor.

Acknowledgment Times for Polling Commands.

Instead of manually creating Z-Wave traffic, the MiCasa Verde VeraLite controller is configured to automatically poll two devices every 60 seconds. The two devices being polled are the FortrezZ water-valve and an Everspring AN145 lamp socket switch. To collect enough frames to be able to measure at least 70 acknowledgments from each of the three devices, the sniffer collects traffic over several hours. Figure 13

shows the 99% confidence intervals estimating the true mean response delays for each device. The figure reveals several interesting things. First, there is a clear difference between the response time of the VeraLite controller and the other two devices. The difference in means between the water-valve and lamp switch is not detectable; however, the data suggests that the lamp switch's sample mean response time has larger variance than the water-valve. Secondly, the mean response times of water-valve for this experiment are different from the experiment summarized in Figure 11. In fact the sample mean variance of all devices in this experiment are larger than those observed in the other experiments. This may be explained by two differences between this experiment and the others. In this experiment there are three devices instead of two. This increases the contention for the RF medium, which increases the variance in delay for when an ACK may be transmitted. For the other two experiments, the medium is always available for a sender because either 1) the previous transaction has already completed, or 2) the other device is waiting for the sender's next transmission. A second explanation of the increased variance is because in this experiment there are 60 seconds between each transaction, where the other two involved repetitions at intervals of one or two seconds. During the period of inactivity, it is reasonable to expect that the devices to go into a low-power state, shutting down several components. The effort to resume an active state must incur additional variance, such as when the internal duty-cycle of each device happens to realize a new transmission is occurring.

Analysis of the Differences in Mean Acknowledgment Response Times.

The experimental data suggest that the differences in means are due to the implementation differences in the device firmware handling acknowledgments. Upon receiving a frame, a checksum is performed. If the checksum passes and the frame

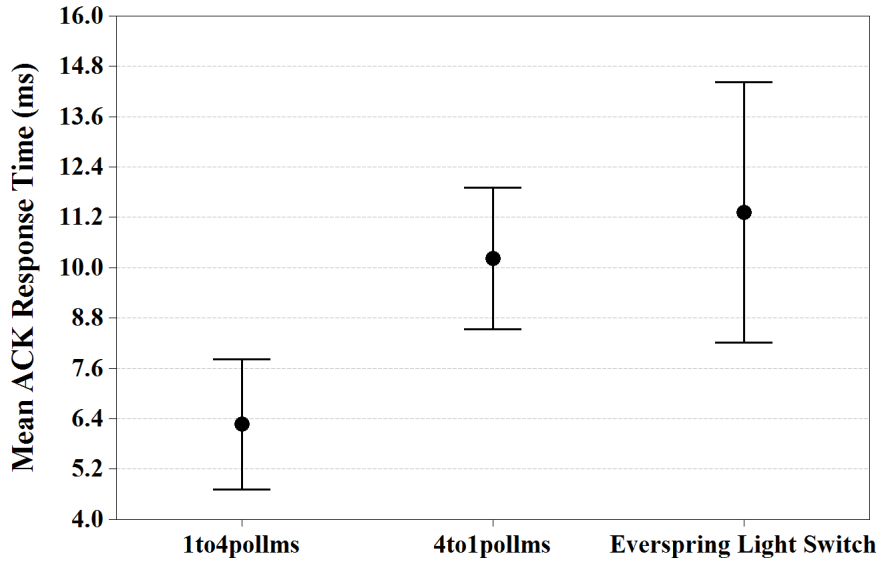


Figure 13. The 99% CI of the ACK response delay at a 60 second polling interval.

has either requested an ACK or the protocol requires it, the recipient generates the acknowledgment using the sequence number of the received frame and sends the response to the originator. The differences arise depending on when these steps are taken within the transceiver chain. The ACK response may be initiated immediately upon receiving the frame or may be queued to avoid interrupting an incoming frame reception. This option, for example, is provided in the CC2420 ZigBee Transceiver [TI15].

After the ACK is generated, another implementation decision involves the transmission of the ACK frame. If the ACK is given priority, it is either placed at the front of the send queue or an interrupt is generated to force the transceiver to immediately transmit the ACK. Other design choices include the sizes of the receive and send queues, buffer exception handling, and the queue service rates. In a delay-prone scenario, ACKs are handled by an attached microprocessor via the Serial Peripheral Interface (SPI) bus.

Another source of differences may be the ZM3102N configuration settings. The

ITU-T G.9959 specification lists parameters relevant to ACK response times such as *MacMinAckWaitDuration*, *TurnaroundTimeRXTX* and *MacMinCCARetryDuration* [ITU12a]. Three parameters specify the minimum frame spacing between receipt of a packet and transmission of its acknowledgment, the time penalty for switching from receive to transmit mode, and the minimum time to wait between clear channel assessments. It is not clear which, if any, of these parameters are configurable on the ZM3102N; however, each may impact the ACK response time.

2.5 Conclusion

This work explores the susceptibility of Z-Wave devices to passive reconnaissance techniques and device fingerprinting. Follow-on work is necessary to more rigorously evaluate these techniques to determine if they hold for the greater population of Z-Wave devices. Additional research is required to understand the true cause of the differences in mean ACK response times. Currently, the most feasible approach is to extract and analyze device firmware to identify the mechanisms dictating acknowledgement response delays, providing the motivation for the firmware extraction and analysis activities conducted in Chapter III.

Beyond passive techniques, future work should invest in pursuing active techniques that probe devices to aid in fingerprinting devices. One active technique could be to exhaustively send all messages for all command classes to a target device to determine a complete understanding of the device’s capabilities. The work reveals the need to develop a tool that incorporates both passive and active techniques to fingerprint devices, with functionality similar to the *nmap* network device fingerprinting tool. Additional security aspects of Z-Wave systems are explored in later chapters, including the network and security layers. The ultimate goal is to explore possible vulnerabilities and develop mitigating strategies. The Z-Wave passive frame sniffing

capability developed herein is extended and utilized extensively in the later chapters.

In this work, the security implications of using Z-Wave devices for IoT applications are explored. Specifically, several techniques to passively discriminate between Z-Wave devices by functionality and vendor are examined. Several experiments are performed for the passive technique of observing ACK response times to identify differences in the firmware of devices using the ZM3102N Z-Wave transceiver. It is observed that a VeraLite controller and FortrezZ water-valve have different mean ACK response times, while an Z-Stick controller and an Aeon Labs Appliance Switch have the same response times, suggesting they use the same firmware. The results show promise, but more research is required to fully understand the implications of this discovery.

III. Extraction and Analysis of Non-Volatile Memory of the ZW0301 Module, a Z-Wave Transceiver SoC

3.1 Introduction

The IoT is just one in a series of steps towards a society that is completely immersed in ubiquitous computing. Ubiquitous computing promises highly available information services, regardless of locality. This requires a high density of integrated processing elements, sensors, and actuators to be embedded within areas of human operation.

The IoT is a technology trend with a growing market [Roh15] to interconnect items that contain embedded computers, such as thermostats, security systems, and appliances. Moreover, the number of networked and embedded domestic *things* is on the rise. While the actuation of a door lock or light switch can be implemented without an embedded processing element, new commercial products are including them to be able to connect with the IoT [Aeo17]. Even non-IoT items may be controlled through proxy IoT devices that supply their power, such as smart power switches and outlets.

An advantage of a connected network of things is centralized command and control; however, this functionality may also be automated. This means that a centralized computing node uses a user-defined ruleset to automatically determine how to respond to given conditions. In addition to automation, the IoT network is typically connected through a gateway node to the Internet. This provides connectivity to smartphones, giving the user the capability to remotely lock a door or monitor power usage outside of their home.

Example IoT applications include home security, smart energy, and safety critical services. An example of the latter case involves a commercially available water leak

monitoring system composed of several water sensors and a water valve. In the event that a water sensor detects fluids, a signal is sent through the IoT network to the water-valve actuator to close the water supply and prevent further water damage [For15].

Z-Wave.

Z-Wave is an implementation of a complete IoT substrate, containing well-defined communication, networking, and application layer protocols [Pae13]. Z-Wave capable sensors, actuators, controllers, routers, and Internet gateways may be combined by the user to provide home automation services. To minimize installation costs and make Z-Wave more competitive, the system is predominantly wireless. Rather than utilize an existing radio protocol stack, such as Wi-Fi or ZigBee, Z-Wave uses its own stack. The physical layer and medium access layer are defined in [ITU12a], but routing and application layer services are not available as a specification in open source literature.

The ability for a vendor to enter the Z-Wave market is controlled by a single company. Originally, this company was Zensys; however, in 2009 it was announced that the company was acquired by Sigma Designs [Hig17], which has continued to follow the same model. This centralized management approach has advantages over other low-power RF protocols, such as ZigBee. In the case of ZigBee, there are many different implementations and some are not compatible. In contrast, a Z-Wave vendor uses the interfaces and protocols specified by the Sigma Designs company, increasing the likelihood that all devices are compatible, reducing the complexity of user installation and operation.

To produce Z-Wave devices, a vendor must coordinate with Sigma Designs and sign a NDA regarding the proprietary details on Z-Wave implementation. Once in

place, Sigma Designs provides the necessary hardware and software to develop and market Z-Wave compatible devices [Sig15]. The majority of all Z-Wave capability is contained on a small postage stamp sized SoC that the vendor is responsible for integrating into their system. Sigma Designs also provides example firmware images, software libraries, and documentation to assist with the software development. Additionally, they provide vendors with a development board and firmware programmer device.

Pairing Operation.

A distinguishing aspect of Z-Wave is the manner that devices enter and leave the network. This *inclusion* or *pairing* process is similar to Bluetooth device pairing. A user wishing to add a device to the network first puts the Z-Wave controller and the new device into pairing mode. While in pairing mode, the controller may add any devices that are also found to be in pairing mode. Depending on the implementation of the controller, it may either automatically join the discovered nodes to the network or require user authorization prior to joining them. Unpairing devices is accomplished in a similar manner.

Node Identification.

Z-Wave devices are identified by a 4-byte *Home ID*, which is assigned by the controller to each device during the pairing process. All nodes paired to a given controller share the same Home ID, which is assigned to the controller by the vendor when the device is fabricated.

The second form of identification is the *Node ID*. A Node ID is a byte value and is also assigned by the controller to a device during the pairing process. The controller node always has the lowest Node ID of 1. The first device paired to a controller has

a Node ID of 2. Subsequent pairing operations result in an assignment of an unused and monotonically increasing Node ID.

Command Class.

Z-Wave networks include controllers, sensors, and actuator devices. To ensure device compatibility, the application layer protocol is well-defined; however, not all devices require the ability to participate in every application layer transaction. For example, a light switch does not need to know how to respond to a request for a temperature reading. Consequently, the application layer is partitioned by functionality into a series of *command classes*. A given Z-Wave device belongs to one or more command classes and the associated application layer protocol functionality is included during compilation of the firmware image. A device announces this set of supported classes to the controller during the pairing operation. Each command class is a unique 8-bit value [Ope15b].

IoT Vulnerabilities.

Since IoT devices interact with people, some will be in the vicinity of a crime, or worse, used as a means of achieving a particular crime. Therefore, forensic analysis of these devices may provide useful evidence in the resulting investigation. For example, Z-Wave systems may be used as a security system for a home or office. It may be useful to ascertain how an intruder acquired access to the facility without tripping the alarm. In one of the earliest investigations of Z-Wave security, a vulnerability is exploited in a Z-Wave door lock device that allows attackers to reset the Z-Wave communication key. With the key reset and known by the attacker, the door can be disengaged remotely [FG13].

A compromised Z-Wave network may also be used by an intruder to conduct

reconnaissance. For example, a distributed system of Passive Infrared (PIR) sensors may reveal patrolling patterns of security forces in a facility or, more generally, the occupancy state of a particular room. More sophisticated sensors may be accessible through devices that bridge between a Z-Wave network and the business or home IP network. Passive collection techniques for performing reconnaissance and device enumeration on Z-Wave networks are explored in [BFH⁺15].

With a demand by users to control everything from everywhere, vendors build Z-Wave controllers that also act as gateways between IP and Z-Wave networks. By compromising the Z-Wave controller, an attacker may gain access to the IP network. In [FR15], the controller replication functionality is exploited on the WLAN side of a home network to take command of a Z-Wave network.

An underlying threat to Z-Wave systems is that physical access to a Z-Wave device allows someone to conduct a *firmware modification attack*. While some devices have tamper protections, such as housing triggers to detect when opened, most are housed in plastic cases that are easy to disarm and access using common tools. The threat to a user is that new firmware, such as one including a rootkit, may be uploaded to the device to alter functionality.

Firmware rootkits are discussed in [SVM15]. While the article focuses on firmware modification of subsystems in desktop computers, the same principles apply to embedded system firmware. In [CCS13], the entire process of conducting a firmware modification attack is elucidated for the firmware of a network printer. Rootkits on Z-Wave devices may be used to collect sensor information, suppress alarms and other actuators, act as a proxy to access the IP gateway, or discretely manage the Z-Wave devices.

Related Work.

Early Z-Wave device forensics analysis is provided by Jean-Michel Picod [Pic14], wherein the steps on extracting the firmware from Z-Wave chips are detailed. Through this process, software is developed and made available for use on the GoodFET [Goo15]. Picod et al. also provide a SDR implementation of a Z-Wave transceiver [PLD14].

Beyond the work by Picod, forensic analysis techniques for Z-Wave devices have not yet been explored; however, research has been done on other IoT devices. A forensic analysis of a Smart TV is performed in [BRR15], where the authors provide several memory extraction techniques analogous to Z-Wave devices. A PlayStation 4 is analyzed for digital artifacts in [DRXS15].

There are several other open source Z-Wave tools available. OpenZWave is an open source Z-Wave library to facilitate controller behavior operating on a PC [Ope15b]. Under the Github group AFITWiSec [AFI15], several Z-Wave tools are provided. These include a Wireshark dissector for Z-Wave MAC and network layers, used in [BFH⁺15], and a Z-Wave auditing tool.

Regarding Z-Wave security, preliminary work is conducted to use RF features to authenticate the Z-Wave devices in [BBTR15] and [PR15]. An RF fingerprinting capability on Z-Wave devices enables mutual authentication between devices and the controller. More generally, there have been some efforts regarding the security of abstract IoT systems. A favorable summary of trust management frameworks for IoT is provided in [YZV14]. In [NGFB15], the authors adapt a model-based framework to the IoT security (i.e., trust management) problem.

Contributions.

From a forensics perspective, a law enforcement agency may not want to be bound by a NDA, especially if the information protected by the NDA is evidence in a case. Moreover, it may not be legal for a government agency, such as a United States federal agency, to be bound to a contract under a different legal system. To avoid these issues, forensic investigators may turn to open source literature on what is currently known about Z-Wave systems to aid in their investigations. Therefore, the intent of this chapter is fourfold. The contributions of this work are to:

1. Characterize the Z-Wave hardware and software architectures.
2. Provide methods of exfiltrating data from Z-Wave hardware memories to enable future forensic investigations.
3. Investigate how Z-Wave applications utilize memory.
4. Identify several memory artifacts useful for digital investigations.

The authors have not signed a NDA with Zensys or Sigma Designs. Consequently, all information provided within this chapter is derived from the results of this research effort or as cited.

The remainder of this chapter is organized as follows. Section 3.2 provides background material for the internals of a Z-Wave device. Section 3.3 demonstrates that a firmware modification attack is possible for Z-Wave devices. Section 3.4 imparts the extraction and analysis tools and techniques used for this effort. The analysis results of the collected firmware images are detailed in Section 3.5. Section 3.6 identifies the data structures and artifacts discovered in the acquired Electronically Erasable/Programmable Read-Only Memory (EEPROM) images. The chapter ends with the conclusion and future work in Section 3.7.

3.2 System Under Analysis: ZW0301

The target chipset for this research is the Zensys ZW0301, which is found in the Zensys ZM3102 SoC package. Approximately the size of a postage stamp, the ZM3102 is commonly found to be attached to the mainboard of many commercially-available Z-Wave devices.

ZM3102 Module.

The ZM3102 is a hardware module that provides a Z-Wave transceiver, RF front-end, and RF filter; all of which are shown in Figure 14. Shown in the upper left of the module, the Zensys ZW0301 is a proprietary Z-Wave transceiver and an extended form of an Intel 8051 Micro-Processor Unit (MPU) [Zen07]. The ZW0301 non-volatile memory hosts the Z-Wave network stack, a proprietary Z-Wave API handler, and contains address space to host device-specific behavior on the chip. This avoids the need for a device vendor to require an additional MPU and external memory for their application [Zen07].

The RF front-end component is located in the lower right side of the module. While details about the front-end are not available, a RF front-end is typically used to perform up and down conversion between RF and base-band frequencies. The lettering on the RF front-end module indicates if the transceiver uses United States (US) or European (EU) frequencies. Located in the upper right near the two antenna-pin connectors is the front-end RF filter, which lies between the RF front-end circuit and antenna on the signal path.

ZW0301 Architecture.

The ZW0301 contains internal RAM, extended RAM, a Flash memory, application specific I/O, and an 8051 core [Zen07]. Because of the 8051 core, the MPU is a

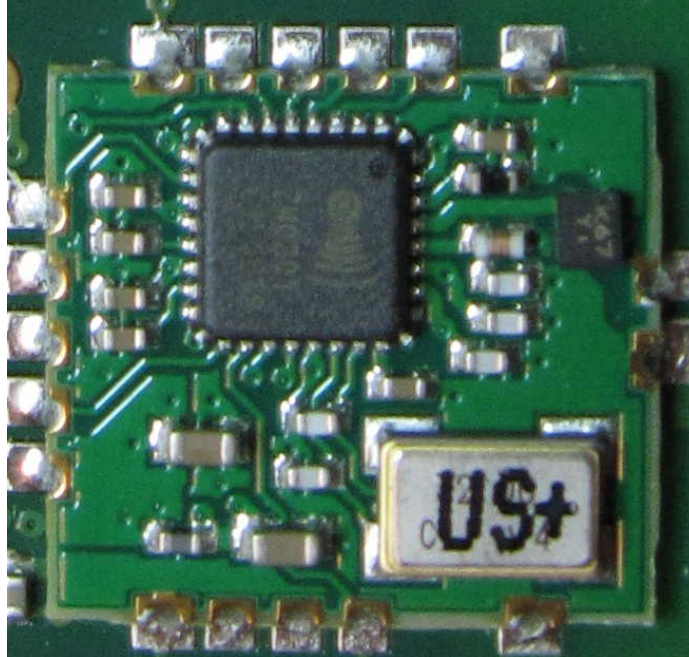


Figure 14. ZM3102 module with ZW0301 Z-Wave transceiver, RF front-end circuit, and RF filter.

Harvard Architecture, where the code and constant data exists in non-volatile memory that is physically distinct from RAM.

Intel 8051 Architecture.

One advantage of the 8051 is that it is widely available from many manufacturers and comes in multiple forms. This also makes defining a generalization of the 8051 architecture difficult. From [Ste05], the maximum addressable memory is 64 KB, which can be expanded through memory banking. A clock-speed of 11 MHz is also common for 8051 devices. With respect to I/O, the basic 8051 provides at least four I/O ports, two external interrupts, two 16-bit timers, and a serial port [Ste05].

The processor contains a small amount of byte-addressable working RAM that is synonymous with its register set. This internal RAM block is segmented into several regions. Four general purpose register banks exist at the lower end of the RAM. While the memory is byte-addressable, there is a bit-addressable region above the register

banks used for storing application specific flags. This is followed by a segment of common RAM that extends to the middle address-space of the RAM. The upper half of the RAM is reserved for Special Function Registers (SFRs).

In the 8051, a SFR is simply a name given to a particular byte address within the internal RAM block to denote implicit internal or external usage by the MPU [Ste05]. For example, the Serial Buffer (SBUF) register is located at RAM memory address 0x99. When a new byte arrives at the MPU over the serial port, the MPU will store the byte in SFR SBUF to be processed by application code. With respect to the original Intel 8051 design, there are 21 SFRs in the upper memory to support I/O, interrupts, timers, condition flags, and maintain the Program Counter (PC) [Ste05]. The remaining SFRs are available for 8051 manufacturers to provide additional capabilities beyond the original design.

The Intel 8051 shares its instruction set with the 8052 MPU [Ste05], where each instruction is one to three bytes. The 8051 performs arithmetic and logical operations on 8-bit values. To address up to 64 KB, two 8-bit registers may be utilized for load, store, and jump instructions. For example, the DPTR register is a 16-bit register composed of DPH and DPL 8-bit registers. Several C compilers exist for this target including the Small Device C Compiler (SDCC) for Linux [Jar15] and the Keil C51 compiler [Kei15b] for Windows. The assembly language for the 8051 is defined in [Met96].

Memory structure.

The 8051 core of the ZW0301 contains a 256 byte internal RAM block. The ZW0301 provides a 2 KB extended RAM block and 32 KB Flash memory [Zen07]. The 32 KB Flash memory can be reprogrammed externally using an SPI bus when the ZW0301 is placed into reprogramming mode [Zen07]. While not found in all Z-Wave

devices, an external 8-pin Small-Outline Integrated Circuit (SOIC) EEPROM may be found on the device mainboard adjacent to the ZM3102 package. The ZW0301 uses the SPI bus to access the external EEPROM, which provides persistent data storage.

Input / Output.

The ZW0301 design satisfies several Z-Wave applications by providing multiple types of I/O interfaces [Zen07]. For sensing applications, the ZW0301 provides an 8 or 12-bit Analog-Digital Converter (ADC). A triac controller is provided for smart energy applications. Interfaces for Universal Asynchronous Receiver/Transmitter (UART) and SPI are also provided to allow the package to be integrated with existing systems and to reprogram the transceiver MPU.

Z-Wave API.

The Z-Wave API provides a command and control interface to the ZW0301 so that a host device may act as a controller through the ZW0301. While Z-Wave controllers are found to respond to the serial API protocol messages, it is observed that each controller may respond to a device-specific subset of the known API calls.

OpenZWave is an open source library developed for USB Z-Wave controller modules. While Z-Wave is closed source, OpenZWave reveals details about the opaque serial API for the ZW0301. Analysis of the OpenZWave source code shows that the API protocol is asynchronous and transaction-based. A transaction involves the host issuing an API call as a structured message over a serial interface connected to the ZW0301. API calls are identified by an 8-bit index value included in the fourth byte of the frame. Known API call indexes are defined in *defs.h* in [Ope15a] of OpenZWave source code.

3.3 Z-Wave Firmware Modification Attacks

In Section 3.1, the firmware modification attack is identified as a vulnerability of Z-Wave devices using the ZW0301. An example of firmware modification is demonstrated for an Ecolink PIR sensor. Several instructions in the firmware image are modified at the start of the *main* C function, while not changing the behavior of the system.

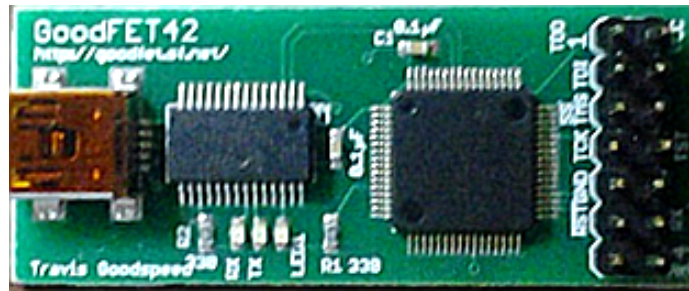


Figure 15. GoodFET 42 board used for ZW0301 reprogramming.

It is assumed that a system failing an integrity check will go into an error state, where it is not executing application code. In the case of the PIR sensor, the device will cease reporting detection events over RF. On the contrary, observing a device reporting motion detection events to a controller over RF after the firmware image is modified implies that the modified instructions have been executed. In the event of the latter case, it is shown that the ZW0301 does not detect unauthorized modification of firmware images.

```
main:                                ; 3 bytes
mov     DPTR, #0x7FA2
clr     A                             ; 1 byte
movc    A, @A+DPTR                    ; 1 byte
mov     ZW0301_CF, A                  ; 2 bytes
lcall   Toggle_C2_b2_clear_b3
```

Figure 16. Disassembly of the main function in the firmware of an Ecolink PIR sensor.

The firmware modification is accomplished by first retrieving the firmware from

the device using the GoodFET, shown in Figure 15, using the process described in Section 3.4. The location of the main function is found at address 0x0649 using the approach revealed in Section 3.5. Figure 16 shows the disassembly of the beginning of the main function. When executed, the instructions load the byte value stored at address 0x7FA2 in Flash memory to internal RAM address 0xCF.

```

main:                                     ; 3 bytes
mov    DPTR, #0x7FA2
mov    A, #3                             ; 2 bytes
mov    ZW301_CF, A                       ; 2 bytes
lcall  code_685D

```

Figure 17. Disassembly of the modified main function in an Ecolink PIR sensor firmware image.

Examining the constant data section of the extracted firmware image reveals that 0x03 is stored at address 0x7FA2. Since this firmware value is invariant, directly loading immediate value 0x3 is equivalent to retrieving it from the firmware. The instruction to move an immediate to accumulator register A requires two bytes. To avoid changing the size of the binary, the CLR A and MOV A, @A+DPTR instructions are replaced with MOV A, 0x03. With respect to the byte values, the two original single byte instructions 0xE4 and 0x93 are respectively replaced with a single two byte instruction 0x7403. For this particular PIR sensor, there are no side effects to this modification because the DPTR and A registers and RAM address 0xCF all have the correct states prior to the LCALL function call. After modifying the firmware image, it is disassembled in IDA pro and is shown in Figure 17, which contains the intended instruction.

The modified firmware image is written to the ZW0301 using the GoodFET. The memory writing tool included in the GoodFET software package does not erase the memory prior to writing, so this must be performed manually to correctly write a new image. In addition to reading and writing, the tool also includes the ability to

```
Verifying code from 000000 to 007fff as ./ecolink  
Verified 000000.  
Verified 000100.  
Verified 000200.  
Verified 000300.  
Verified 000400.  
Verified 000500.  
Verified 000600.  
000652: e4/74  
000653: 93/03  
2 bytes don't match.  
Verified 000700.
```

Figure 18. Verification that the firmware of the Ecolink PIR sensor is modified when compared to the factory firmware image.

compare a file to the firmware memory on the device, which is used to verify that the Flash memory is updated correctly. Figure 18 shows that, in the ZW0301 of the PIR sensor, address 0x0652 is changed from 0xE4 to 0x74 and the byte at 0x0653 is modified from 0x93 to 0x03 when compared to the original firmware image.

To attempt to observe Z-Wave frames over RF, the device is returned to normal operation. The GoodFET jumper wires are removed, the battery is inserted, and the cover is replaced to depress the tamper switch. A Z-Wave frame sniffer consisting of a B210 Ettus SDR, Scapy-Radio Z-Wave gnuRadio module, and a custom Z-Wave frame dissector for Wireshark are used to observe RF communication between the PIR sensor and an Aeon Z-Stick2 controller. The results show that the PIR sensor exhibits normal behavior; it continues to report motion detection events to the controller when stimulated in the sensed phenomenology. Consequently, a modified firmware image is running on the PIR sensor.

Implications of Remote Programming.

While it is certainly possible to reprogram the behavior of the system given physical access, an open question is if it is possible to change the execution behavior remotely. It is observed on some systems, such as the Ecolink PIR sensor, that the data and memory share the same address space. Such a design makes the system

more susceptible to remote code exploitation vulnerabilities. This is especially true when a remote device controls what is written to memory. For example, the Ecolink PIR sensor writes the Home ID to several locations in Flash memory upon a pairing operation. This provides a potential attack vector, where at most four machine instructions may be embedded in the addresses reserved for the Home ID. While this is limited and still requires a program counter redirection to execute, not enough is known about the pairing operation to determine if more data may be written, either legitimately or illegitimately, to Flash memory during this event. Such a capability would make devices even more susceptible to remote code exploitation.

3.4 Methodology

From Section 3.2, there are four memory regions of the ZW0301 that may contain digital forensic evidence. These include the internal RAM, extended RAM, Flash memory, and external EEPROM. For this work, the Flash and EEPROM memories are acquired and analyzed. RAM image acquisition and analysis are left for future work.

Flash Memory Extraction.

From [Mot03], the Master-Input Slave-Output (MISO), Master-Output Slave-Input (MOSI), SPI Clock (SCK), Not Reset (RESET_N), V_{cc} , and Ground (GND) pins of the ZM3102 are needed to facilitate a SPI transaction. The pin-out diagram of the ZM3102 package is provided in [Zen07], revealing that the SPI pins of the ZW0301 are made external to the ZM3102 package at pins 7, 9, 8, 2, 11, and 1 respectively. The ZM3102 package may be quickly oriented by identifying the edge of the module that has only two pins. These are pins 18 and 19, which interface to the external antenna on the mainboard of the device.

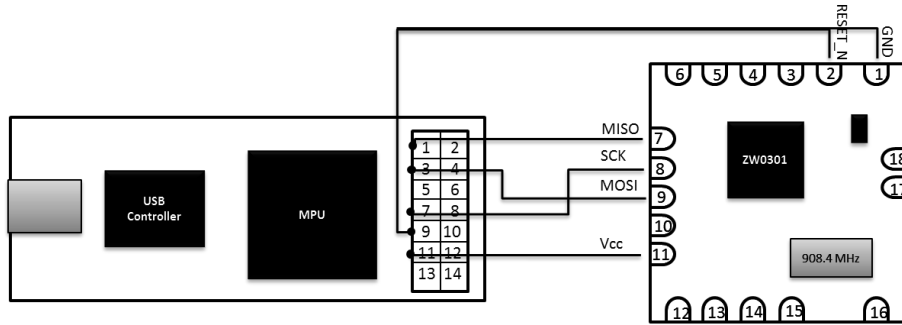


Figure 19. GoodFET to ZM3102 interface block diagram.

Figure 19 is a block diagram, derived from [Goo15], of the pins that must be connected to dump the Flash memory. Depending on the device manufacturer, headers or test pads may be provided on the mainboard for these pins. If these are not provided, then jumper wires must be soldered to the pads connecting the ZM3102 to the mainboard. In some cases, additional solder must be added to the pad to adequately support a jumper wire. The edge of the ZM3102 containing the MISO, MOSI, SCK, and Vcc pads is especially troublesome. Because of the mutual proximity of these pins, it is easy to cause solder bridges using common soldering equipment. Moreover, the manufacturer may place large components around the ZM3102, making it physically difficult to access the pads connected to the package.

```
root@kali:~# export board=goodfet42
root@kali:~# goodfet.zensys info
Jedec: 7f 7f 7f 7f 1f
Chip type: 00
Chip revision: 07
Identified ZW0301 chip
root@kali:~# goodfet.zensys dump ./toodlesMcFly.rom
Dumping code from 000000 to 007fff as ./toodlesMcFly.rom.
Dumped 000000 -> 000100.
Dumped 000100 -> 000200.
Dumped 000200 -> 000300.
Dumped 000300 -> 000400.
```

Figure 20. Sample output of the goodfet.zensys info and dump commands.

After connecting the soldered leads on the ZM3102 to the GoodFET board and connecting the GoodFET to a PC via USB, the extraction software tool is invoked. A python script named *goodfet.zensys* is included in the GoodFET software pack-

age, which is capable of reading, validating, and writing the ZW0301 Flash memory. Figure 20 shows how the ZW0301 Flash memory is retrieved using this tool.

External EEPROM Extraction.

Dump-Zprom.

Dump-Zprom is a tool developed to utilize the *Read Memory* serial API call [Ope15a] to retrieve memory from the ZW0301 over the serial port. While the name given to this API call is ambiguous as to which memory is being read, it can be deduced. Reading congruent memory addresses reveals the firmware image to be largely vacant with a few data items scattered over the memory. Moreover, the addressable range appears to be from 0x0000 to 0x3FFF, which is larger than the available address space for the extended RAM and internal 8051 RAM. By deduction, this leaves the external EEPROM as the target of the API call.

Dump-Zprom is a command-line tool developed by the authors for the work herein. It requires a path to the serial device file as its only argument. The tool is hard-coded to read all bytes from 0x0000 to 0x3FFF, read 16-bytes per API transaction, and print the results to standard output.

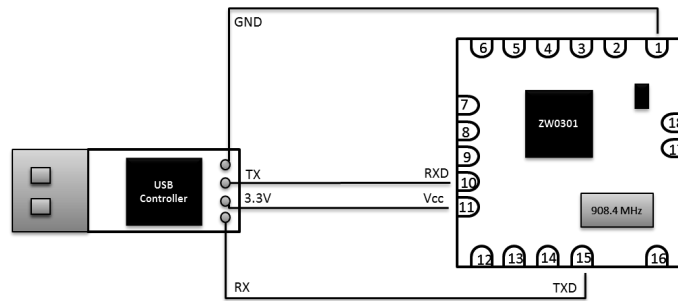


Figure 21. USB controller to ZM3102 interface block diagram.

To use the Dump-Zprom tool, the ZM3102 must be connected to a USB controller. If a Z-Wave device lacks a USB controller, a USB breakout cable may be used. Figure 21 shows a diagram of the interface between the ZM3102 and a USB controller

with a USB Type A connector. To avoid damaging Z-Wave devices, the 3.3 V power-supply pin must be used instead of the 5.0 V power pin. The break-out wires from the USB controller are soldered to the ZM3102, where the TX and RX pins are crossed between devices. Once the soldering is completed, the USB breakout cable may be plugged into a PC to execute the Dump-Zprom tool on the serial interface assigned to the break-out cable.

EEPROM Reader.

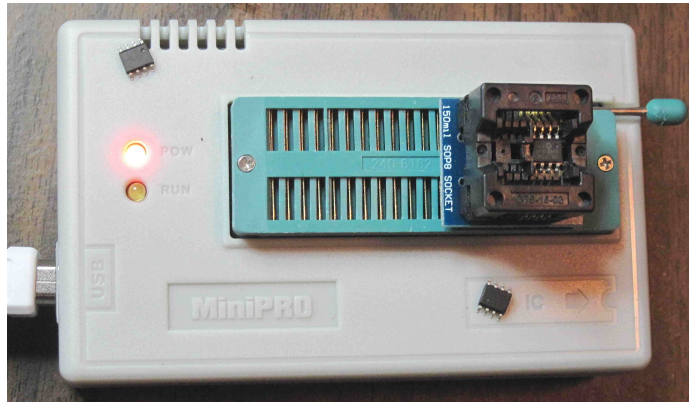


Figure 22. MiniPRO EEPROM programmer with SOIC8 to DIP8 converter reading a Micron 25LC256 EEPROM.

In the event that a target ZW0301 does not service memory read requests over its serial API, an EEPROM programmer may be used. For example, a MiniPRO EEPROM programmer is used to confirm that the Dump-Zprom tool is correctly acquiring the EEPROM memory. To read the target EEPROMs, a SOIC8 to DIP8 converter is necessary. Figure 22 shows a MiniPRO, DIP8 converter, and a Micron 25LC256 EEPROM, which is desoldered from a LynxTouch module.

The tool comes with a Windows program and a repository of EEPROM and MPU chip specifications. After selecting a target IC type, the user may read, write, or verify data on the chip. Figure 23 shows a portion of the memory image extracted from a 25LC256 Microchip 256 Kbit EEPROM [Mic05], which is removed from a

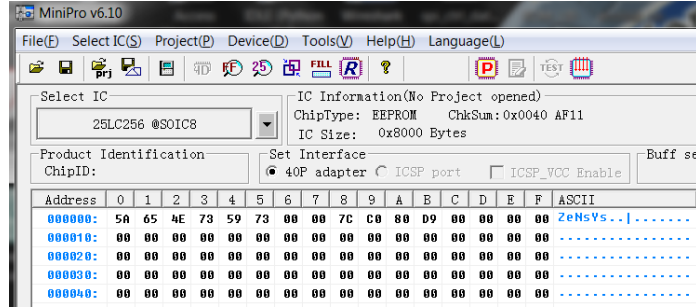


Figure 23. MiniPRO software displaying the contents of a Micron 25LC256 EEPROM.

Honeywell LynxTouch Z-Wave controller module. A drawback of this method is that the EEPROM must be isolated; however, desoldering the IC is highly suggested. Note: Several EEPROMs have been damaged while attempting to utilize the SPI pins of the EEPROM while attached to the device mainboard.

Flash Memory Analysis.

Firmware images of the ZW0301 Flash memory of an Ecolink PIR sensor, Trane thermostat, Linear garage door controller, and LynxTouch add-on Z-Wave module are acquired using the GoodFET tool. These devices are shown in Figure 24. To observe any dynamic aspects of the Flash memory in these devices, they are paired to and unpaired from an Aeon Z-Stick2 controller with a Home ID of 0x0184E0B6. Between events a new firmware image is acquired and compared with the other images. The firmware images are analyzed to identify code and data segments. This is accomplished by using an IDA Pro 8051 disassembler and common hexdumping tools such as *xxd* for Linux.

While IDA Pro includes a disassembler for the 8051 instruction set, it supports a small subset of 8051 configuration variants. None of the included variants match what is known in open source literature about the ZW0301, such as the number of interrupts, timers, I/O ports, and memory sizes. This is an issue because the lower end of memory holds the interrupt and timer handlers in fixed width memory

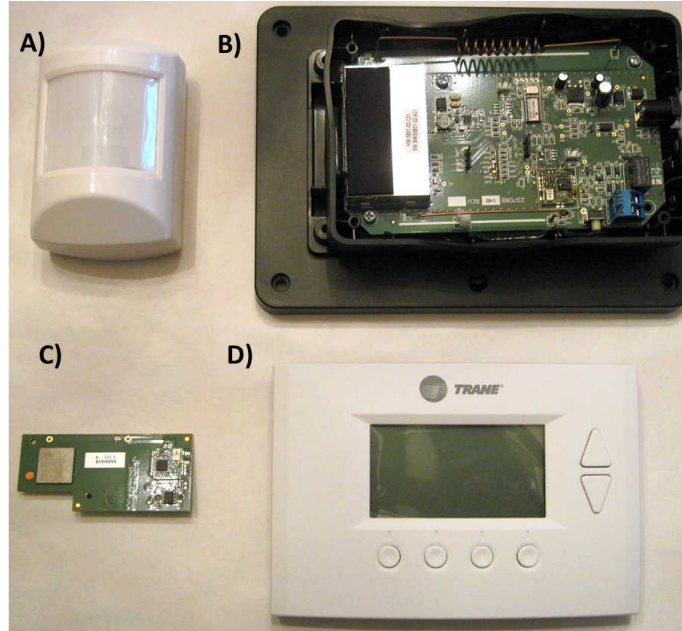


Figure 24. Z-Wave devices with successful Flash memory exfiltration - A) Ecolink PIR sensor, B) Linear garage door controller, C) Honeywell LynxTouch module, and D) Trane thermostat.

segments [Ste05]. Given that IDA Pro does not know the correct numbers of handlers, it disassembles the lower memory region incorrectly, requiring manual intervention. Moreover, the binary image is void of information that would be useful to determine object code segments and where constant data may be located. IDA Pro does not appear to recognize a file header or object metadata in the firmware, making it difficult to determine the Object Module Format (OMF), such as MCS-51 [Int82]. Consequently, it is difficult to determine which sections of the image are correctly disassembled using IDA Pro.

External EEPROM Analysis.

Using a custom-built tool, *Dump-Zprom*, the EEPROM memory of an Aeon Z-Stick2 controller and Vera-E controller are collected. Since the LynxTouch module does not respond to the serial memory read API call, the EEPROM images for this device are collected using an EEPROM reader. The devices are shown in Figure 25.

To identify data items, multiple images of the Z-Stick2 EEPROM are collected during a sequence of pairing and unpairing operations with other Z-Wave devices. Analysis of the EEPROM is performed using a text editor and hex dump tool.

3.5 Results of Flash Memory Analysis

Four distinct memory regions are identified after analyzing the firmware images, which include code/data, configuration data, optional data, and free memory segments. Table 1 provides the sizes of each segment for each image. The marbled code and data section consumes approximately 90% of the available 32 KB memory.

Table 1. List of salient features discovered in Flash memory.

Device	Code/Data	Opt. Data	Config	Free
PIR Sensor	26.6 KB	3.6 KB	128 B	1.7 KB
Thermostat	26.8 KB	3.7 KB	128 B	1.4 KB
Lynx Module	31.7 KB	NA	128 B	172 B
Garage Cont.	31.1 KB	NA	128 B	794 B

To further clarify the firmware organization, Figure 26 shows a model of the Flash memory as observed within the four Flash firmware images. The code portion of the firmware, shown in black, occupies the lower memory region and is aligned with address 0x0000. Data regions are mixed within the code segment and vary in quantity, location, and length with respect to the four firmware images. These regions are revealed by having IDA Pro trace the execution path through the firmware image from address 0x0000. The tracing algorithm within IDA Pro is unable to trace jump addresses that are dynamically calculated in the 8051 code; therefore, it is possible that some of these regions are actually library calls.

Unlike the constant data regions within the code segment, there is a constant data region from 0x7F80 to 0x7FFF that is consistent in location, size, and data for all firmware images. Moreover, analysis of the code reveals data in this region being

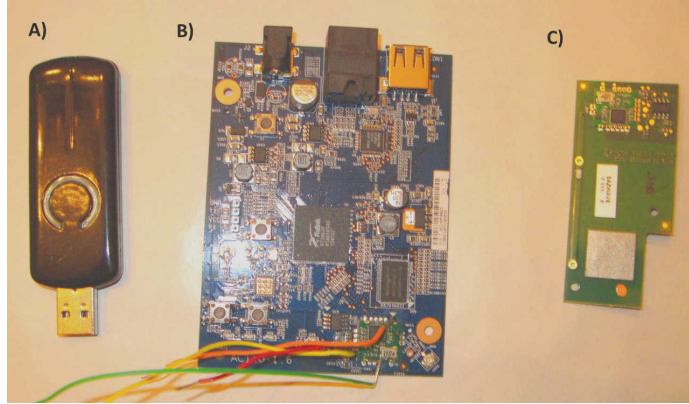


Figure 25. Controllers used in study - A) Aeon Labs Z-Stick2, B) Vera-E controller, and C) Honeywell LynxTouch module.

transferred to external RAM during initialization, confirming that it is indeed a data segment holding configuration information. The portion of the firmware shown in light gray is an optional data segment that is observed in only two of the four firmware images. These data segments are consistent with the reported Flash memory page size of 256 bytes [Pic14]. The remaining memory is free and is shown in white in Figure 26.

The following subsections describe the salient features of the firmware images.

C Format Strings as Constant Data.

Analysis of the Trane thermostat firmware image reveals constant *printf* format strings stored as constant data. Figure 27 shows a portion of this memory segment of the Trane thermostat firmware image. Two examples are outlined in black boxes within the figure. In standard C string I/O, each ‘%’ token indicates a variable is to be inserted at the position of the token. The letters following this token indicate the variable type and other formatting options. The standard I/O function *printf* combines the format string with the variables to construct an output string.

The boxed examples in Figure 27 contain the tokens ‘%s’ and ‘%u’, which are placeholders for a string and unsigned integer respectively. The existence of these

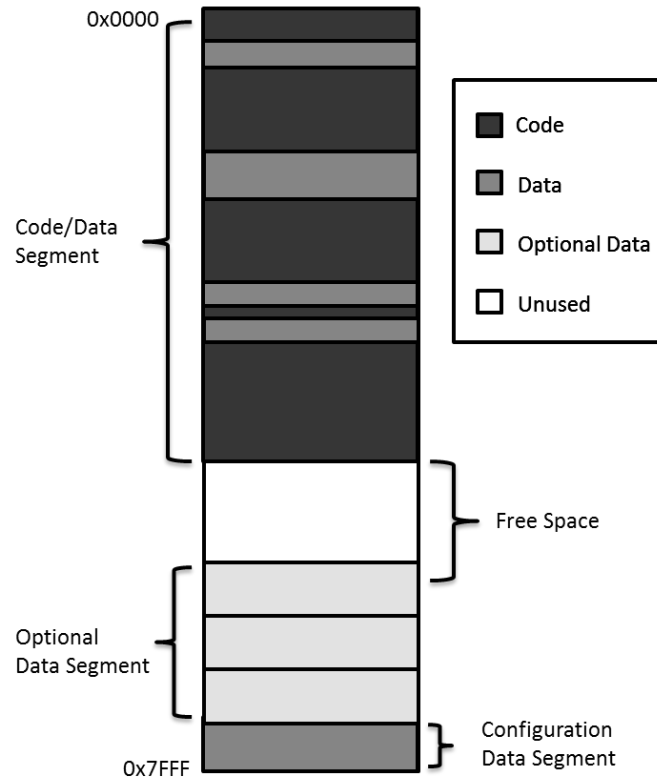


Figure 26. ZW0301 firmware image layout.

strings imply that at least some portion of the firmware is compiled in the C language and statically linked with the *stdio* C-library.

Firmware built with Keil C51 Compiler.

Given that the C strings identify the binary as compiled C code, the exact compiler is fingerprinted by analyzing the initialization sequence of instructions originating from address 0x0000, which is the value of the program counter when the device is initially powered. The evaluation edition of Keil μ Vision 5 provides a macro feature that allows a programmer to declare particular *macro directives* to be used at compile, assembling, and linking for the C51, A51, and L51 tools respectively. The A51 macro assembler provides several initialization directives, which are blocks of assembly code that are inserted into the compiled image automatically. The assembly blocks are

```

3000860: 300d 0025 7353 423d 2562 6400 413d 3235 0. [%sSB=%bd], A=25
3000870: 3520 5350 003d 2562 7500 413d 3235 3520 5 SP,=%bu, A=255
3000880: 4d3d 0041 3d32 3535 2046 3d00 2573 5343 M=, A=255 F=,%sSC
3000890: 3d25 6275 0025 7353 423d 2562 7500 2573 =%bu,%sSB=%bu,%s
30008a0: 5356 2562 753d 2562 7500 2573 5254 463d SV%bu=%bu,%sRTF=
30008b0: 2575 0025 734d 5446 3d25 7500 2573 5254 %u,%sMTF=%u,%sRT
30008c0: 433d 2575 0025 7352 5448 3d25 7500 2573 C=%u [%sRTH=%u],%s
30008d0: 4d54 4d3d 2575 0025 7353 4525 6275 2f25 MTM=%u,%sSE%bu/%
30008e0: 6275 3d00 2530 3262 7525 3032 6275 2530 bu=,%02bu%02bu%0
30008f0: 3262 7525 3032 6275 0041 3d32 3535 2052 2bu%02bu, A=255 R
3000900: 5446 3d3f 0041 3d32 3535 204d 5446 3d3f TF=? , A=255 MTF=?
3000910: 0041 3d32 3535 2052 5443 3d3f 0041 3d32 , A=255 RTC=? , A=2

```

Figure 27. Printf format strings in Trane ZW0301 Flash memory.

provided in the LIB directory of the C51 folder. These files are named INIT.A51, INIT_MX.A51, and INIT_TNY.A51 [Kei15a]. With the exception of jump addresses, the instructions contained in the file INIT_MX.A51 are identical in value and order to what is found in the initialization code for each firmware image. This confirms that the firmware images are built using Keil C51 tools.

INIT_MX.A51 Variable Initialization Macro.

The variable initialization macro provided by Keil is a valuable aid in reverse engineering the firmware code because it labels several jump points. At the end of the macro block, the code jumps to the label corresponding to the *main* function entry point of the firmware. Aligning this block with the disassembled firmware allows the main function to be pinpointed within a firmware image.

```

XDATA: XRAM_0454[4]={0x00,0x00,0x00,0x00}
XDATA: XRAM_04EA[1]={0xFF}
XDATA: XRAM_04EB[2]={0xFF,0xFF}
XDATA: XRAM_04ED[1]={0xFF}
XDATA: XRAM_04EE[1]={0x00}
XDATA: XRAM_0513[1]={0x00}
XDATA: XRAM_0515[1]={0x00}
XDATA: XRAM_0555[1]={0x06}
XDATA: XRAM_056B[2]={0x00,0x00}
root@kali:~/reveng# █

```

Figure 28. Program readINITSEC decoding the C_INITSEC of the Ecolink PIR sensor.

The variable initialization macro also labels the C_INITSEC segment, which is one of the first data segments embedded in the code segment. From [Kei15a], this

section contains the initial values of all static and global variables. Moreover, meta-data is also included in the segment to identify the location and sizes of variables in memory. Unfortunately, IDA Pro is not able to parse this information. A small program *readINITSEC* is developed to decode the C_INITSEC table using information provided in the comments section of the INIT_MX.A51 file to provide details of variable locations, sizes, and values.

The program *readINITSEC* requires the path to the memory image file and the location of the C_INITSEC table relative to the beginning of the file. Figure 28 shows a screenshot of decoding the C_INITSEC of the Ecolink PIR sensor. Each line corresponds to one of the static or global variables. It indicates the RAM or XRAM (i.e., extended RAM) address of the first byte of the variable, the length in square brackets, and the values in curly braces. For example, the first line in the figure reads that a 4-byte value is stored in XRAM at address 0x0454 and is initialized to zeros. This tool will be useful when the RAM and XRAM usage is studied in future work.

Optional Memory Data Blocks.

The PIR sensor and thermostat firmware images contain a region of data blocks. Analysis shows these blocks to be 128 bytes in length and aligned to the start of the configuration data segment at the top of memory, which is also 128 bytes in size. The blocks store a variety of Z-Wave frames used for RF communications.

Figure 29 shows three of the blocks for the Ecolink PIR sensor. The PIR device is paired to an Aeon Z-Stick2 with a Home ID of 0x0184E0B6. The Node ID of this device is 0x04. From [ITU12a], the Z-Wave frame header contains at least the following nine bytes:

1. Home ID (4 bytes)
2. Source Node ID (1 byte)

Block 4	0007d80:	0001	84e0	b6b1	84e0	b604	0304	0a01	0000	Singlecast Frame
	0007d90:	0000	0000	0184	a0b6	0441	0d0d	0100	0000	
	0007da0:	0000	0000	0000	0000	0000	0000	0000	0000	
	0007db0:	0000	0000	0000	0000	0000	0200	0000	0109	
	0007dc0:	0000	0106	f400	0000	0000	0000	0000	0000	
	0007dd0:	0000	0000	0000	0000	0000	0000	0000	0000	
	0007de0:	0001	0101	8202	9520	08cd	0d00	0000	0001	
	0007df0:	0140	0101	36fe	0009	0001	01c5	0101	0100	
Block 3	0007e00:	0001	84e0	b604	54a5	01fe	0000	0003	0000	Low Speed Singlecast Frame
	0007e10:	0000	0000	0000	0000	0000	0000	0000	0000	
	0007e20:	0000	0000	0000	0000	0000	0000	0000	0000	
	0007e30:	0000	0000	0000	0000	0000	0000	0000	0000	
	0007e40:	0000	0000	0000	0000	0000	0000	0000	0000	
	0007e50:	0000	0000	0000	0000	0000	0000	0000	0000	
	0007e60:	0000	0000	0000	0000	0000	0000	0000	0300	
	0007e70:	0000	0000	ffff	ffff	ffff	ffff	ff02	5a7d	
Block 2	0007e80:	0001	84e0	b6b1	84e0	b604	0306	0a01	0000	ACK Frame
	0007e90:	0000	0000	0184	a0b6	0441	020e	0100	0000	
	0007ea0:	0000	0000	0000	0000	0000	0000	0000	0000	
	0007eb0:	0000	0000	0000	0000	0000	0200	0000	0300	
	0007ec0:	0000	0106	f400	0000	0000	0000	0000	0000	
	0007ed0:	0000	0000	0000	0000	0000	0000	0000	0000	
	0007ee0:	0000	0000	0000	0000	0000	0000	0000	0001	
	0007ef0:	0140	0101	36fe	0009	0000	0000	0000	0000	
(...)	0007f00:	ffff	ffff	ffff	ffff	ffff	ffff	ffff	ffff	

Figure 29. Three 128-byte data blocks found in an Ecolink PIR sensor firmware image.

3. Frame Control (2 bytes)
4. Length (1 byte)
5. Destination Node ID (1 byte)

From Figure 29, the boxed byte sequences are Z-Wave frame headers. Block 4 contains the header for a singlecast frame. A low datarate singlecast frame is shown in Block 3, and an acknowledgment frame is included in Block 2. The figure also shows two frames with a unknown frame header type. Per [ITU12a], this unknown type belongs to older versions of Z-Wave.

Available Free Memory.

One final aspect of interest is the amount of unused Flash memory, which is a lucrative place for attackers to house malware. The PIR sensor and thermostat devices both have approximately 3 KB free in Flash. For the garage controller and LynxTouch module, less than 1 KB of memory is unused. While this is limiting, it does not prevent an attacker from utilizing memory that is storing legitimate instructions. The attacker may carefully prune out functionality to make space for malware and allocate some

memory resources for spoofing services to remain masked. Small footprint rootkits are known to be feasible. For example in [Lar14], a rootkit is implemented to covertly run on a water valve Programmable Logic Controller (PLC).

3.6 Results of External EEPROM Analysis

Analysis reveals both consistent and differing memory usage among the controllers. The implication of this behavior is that the EEPROM provides long-term storage for both vendor-specific and common Z-Wave data. Table 2 shows a summary of the findings and where they are located in the EEPROM image for each device.

Table 2. Data and artifacts in EEPROM of three controllers.

Artifact	Aeon	Vera	Honeywell
ZeNsYs String	Not Present	0x1400	0x0000
Home ID	0x1908	0x1408	0x0008
Prot. Info Tbl.	0x19F8	0x14F8	0x00F8
Adjacency Tbl.	0x1E80	0x1980	0x0580
Controller Info	0x0000	0x0000	0x2C00
Pairing Log	0x3XXX	Not Present	Not Present

The Zensys string appears to mark the starting address of the Z-Wave specific data region found in all controllers. The Home ID and protocol information table may be found as positive, constant offsets of the starting address of this string. Figure 30 shows an example byte string retrieved from an EEPROM, where the Zensys string and Home ID are highlighted. While the Aeon controller does not contain this Zensys identifier, the relative positions of the Home ID and protocol information table within the EEPROM reveal the starting address of the Z-Wave specific data, which begins at 0x1900.

Each device also has a multibyte entry containing controller information; however, they are located in different regions for each vendor and do not correlate with the *ZeNsYs* string. This implies that the vendor selects the starting location of this byte



Figure 30. Example boundary string.

vector. More information about the last four items in Table 2 are provided in the following subsections.

Protocol Information Table.

A table exists in the EEPROM that changes its state when a device is added or removed from a controller. It is given the name *Protocol Information Table* because OpenZWave handles the same format of information in function *HandleGetNodeProtocolInfoResponse* of source file *Driver.cpp*. When a device is added to a controller, protocol information about the device is stored in a 5-byte fixed-width record that is indexed by its Node ID. When a device is removed, the corresponding 5-byte record is zeroed. At a minimum, the controller stores its inherent protocol information in the table.

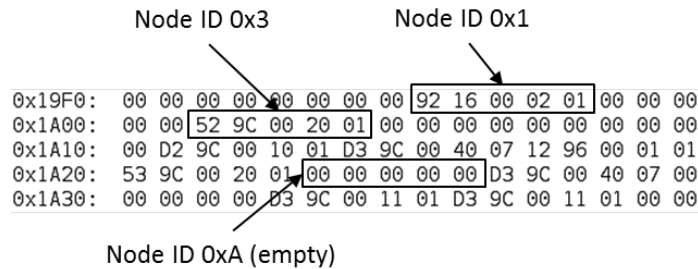


Figure 31. Example protocol information table.

Figure 31 shows a node protocol information table retrieved from an Aeon Z-Stick2 EEPROM. The entries are ordered by Node ID, starting at an ID value of 0x01. The reference boxes in the figure show the entry for Nodes 0x01 and 0x03. Unused IDs have a zeroed entry in memory, such as Node 0x0A, which is also highlighted in the

figure.

For each record, bits 7, 6, and 4 of the first byte are flags that indicate that the device has listening, beaming, and routing capabilities respectively. From Node.cpp of OpenZWave, the second byte holds security flags, and the remaining bytes indicate command classes that the device supports.

Node Adjacency Table.

The adjacency table holds the topology state of the Z-Wave network and is used by the controller to construct routes to devices [rw15]. An adjacency implies that two nodes are able to directly communicate. Including the controller, the largest possible Z-Wave network has 232 nodes [ves15]. The adjacency table has enough bytes to hold the topology state for the largest possible Z-Wave network. The table is located at an offset of 0x0580 bytes from the Zensys string. The table holds 232 records, where each record has a length of 29 bytes. The table consumes approximately 6.7 KB, which occupies over 40% of the EEPROM.

A record in the table corresponds to a particular Node ID, where each 29-byte record contains 232 bits. This allows each record to contain adjacency information for all possible nodes in a network. A node is designated as being adjacent to node i if the i -th bit in the record is set. Unlike other structures in the EEPROM, the records are stored in little-endian format, where the least-significant byte occupies the lowest part of memory.

To illustrate the data structure, Figure 32 shows the node adjacency table from an Aeon Z-Stick2 EEPROM. The first record corresponds to Node 0x01, starting at address 0x1E80. Since the record is little-endian, the byte at 0x1E80 holds the adjacencies for Nodes 0x01 through 0x08. Since bits 7 and 8 are set, then Node 0x01 is adjacent to Nodes 0x07 and 0x08. The byte at 0x1E81 stores adjacencies for

Node ID 0x1 Adjacency Record								
0001e80:	c03b	0000	0000	0000	0000	0000	0000	0000
0001e90:	0000	0000	0000	0000	0000	0000	0000	0000
0001ea0:	0000	0000	0000	0000	0000	0000	0000	0000
0001eb0:	0000	0000	0000	0000	0000	0000	0000	0000
0001ec0:	0000	0000	0000	0000	0000	0000	0000	0000
0001ed0:	0000	0000	0000	0000	0000	0000	0000	0000
0001ee0:	0000	0000	0000	0000	0000	0000	0000	0000
0001ef0:	0000	0000	0000	0000	0000	0000	0000	0000
0001f00:	0000	0000	0000	0000	0000	0000	0000	0000
0001f10:	0000	0000	0000	0000	0000	0000	0000	0000
0001f20:	0000	0000	0000	0000	0000	0000	0000	0100
0001f30:	0000	0000	0000	0000	0000	0000	0000	0000
0001f40:	0000	0000	0000	0000	0000	0001	0000	0000
0001f50:	0000	0000	0000	0000	0000	0000	0000	0000
0001f60:	0000	0000	0000	0000	0100	0000	0000	0000
0001f70:	0000	0000	0000	0000	0000	0000	0000	0000
0001f80:	0000	0000	0001	0000	0000	0000	0000	0000
0001f90:	0000	0000	0000	0000	0000	0000	0000	0000
0001fa0:	0000	0000	0000	0000	0000	0000	0000	0000
0001fb0:	0000	0000	0000	0000	0000	0000	0000	0001
0001fc0:	3000	0000	0000	0000	0000	0000	0000	0000
0001fd0:	0000	0000	0000	0000	0000	0000	0128	0000
0001fe0:	0000	0000	0000	0000	0000	0000	0000	0000
0001ff0:	0000	0000	0000	0000	0001	1800	0000	0000
Node ID 0x9 Adjacency Record								
Node ID 0xD Adjacency Record								

Figure 32. Node Adjacency Table from an Aeon Z-Stick2 EEPROM.

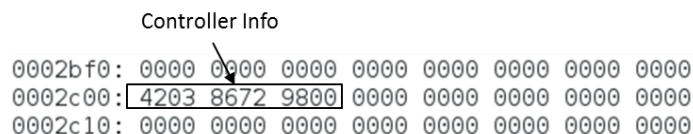
Nodes 0x09 through 0x10. Bits 1,2,4,5, and 6 are set in this byte; therefore, it is surmised that Node 0x01 is adjacent to Nodes 0x09, 0x0A, 0x0C, 0x0D, and 0x0E. The record for Node 0x09 indicates that only Node 0x01 is adjacent. The record for Node 0x0C identifies that it is adjacent to Nodes 0x01, 0x0D, and 0x0E.

Figure 32 also reveals that, for Z-Wave networks having only a few nodes, a significant portion of the adjacency table is not used. The Aeon Z-Stick2 shown in Figure 32 has seven devices paired to it. Including the controller, this consumes only 16 bytes of the 6.7 KB table, leaving 99.8% of the table unoccupied. A large vacancy such as this is a lucrative place for malware data or code to reside.

Controller Information Block.

Figure 33 shows the controller information block at address 0x2C00 for the Lynx-Touch module. With the exception of the first two bytes, each byte is one of the command classes that the controller supports over the RF communication layer. These include Security (0x98), Manufacturer Specific (0x72), and Version (0x86) classes.

Controller Info



```
0002bf0: 0000 0000 0000 0000 0000 0000 0000 0000
0002c00: 4203 8672 9800 0000 0000 0000 0000 0000
0002c10: 0000 0000 0000 0000 0000 0000 0000 0000
```

Figure 33. Controller information within a Honeywell LynxTouch module.

Aeon Labs Z-Stick2 Event Table.

The Aeon Labs controller EEPROM contains a large table that is not present in the other two controllers, indicating vendor-specific behavior. The Z-Stick2 controller differs from the other two devices because it relies on a host CPU that it does not control. The ZW0301 modules for the other two devices appear to be managed through API transactions over a serial line shared with an application processor that is located on the mainboard of the device. In the latter case, the vendor controls both the application processor code and the ZW0301 module. Without control of the host processor, the Aeon controller must place more functionality within the ZW0301 module than the other controllers under study.

A portion of the table is shown in Figure 34. The table is not found at a specific address, but rather, is observed shifting around in memory. Using the Z-Stick may cause the table to move to a new position within the EEPROM; however, this is empirically bounded in the 0x3000 to 0x3FFF address region. The table is composed of 22-byte fixed-length records. Each record corresponds to a particular event, and the table is ordered by event occurrence. Without knowing the precise nature of

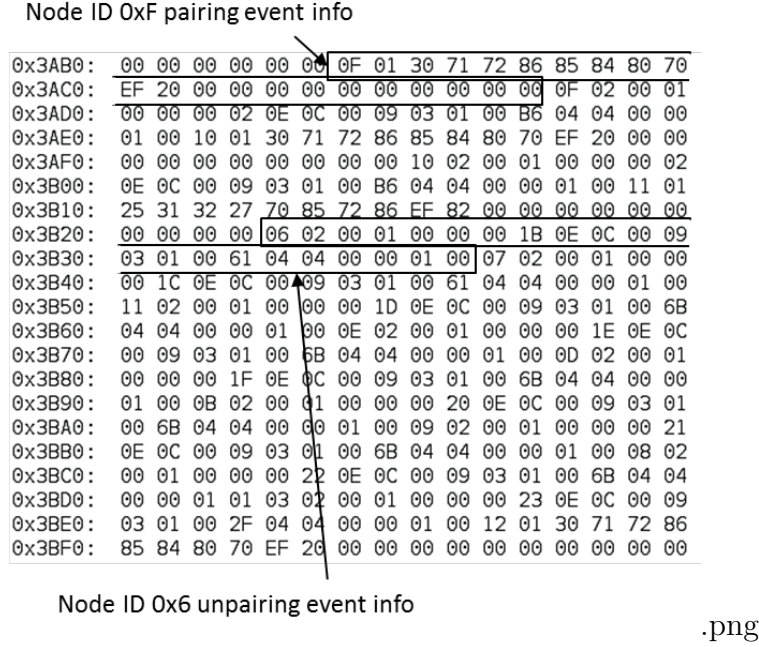


Figure 34. Portion of a Z-Stick2 event table.

the table, the authors designate it as an event table. When devices are paired and unpaired from the controller, there are several counters within this region that track the number of pairing, removals, and size of the event table.

The first byte of an event record is the Node ID of the subject of the event. Events are discriminated by the second byte, where 0x01 indicates a pairing event and 0x02 denotes an unpairing event. For pairing events, the remaining portion of the record is a list of command classes that the device supports. This is confirmed by pairing a FortrezZ water valve to an Aeon Z-Stick2 and examining the event table to see if the expected command classes are listed. The water valve user manual [For15] identifies that the Basic (0x20), Binary Switch (0x25), Manufacturer Specific (0x72), Version (0x86), Configuration (0x70), Alarm (0x71), and Application Status (0x22) are supported by the device.

After pairing the water valve, the EEPROM memory of the Z-Stick2 is extracted and a portion of the event table from the memory is highlighted in Figure 35. The

water valve has a Node ID of 0x19 and, being the most recent pairing operation, is located at the top of the table at address 0x3CF2. The figure shows that, following the Node ID and pairing event bytes, the byte values match what is found in the user manual. Only the Basic command class is not listed.

The data contained in unpairing event records remain unknown. The byte values are generally the same for each record. However, different byte values are observed when a device is removed using the OpenZWave Control Panel versus pressing the unpairing button located on the Aeon Z-Stick2.

With respect to a forensic investigation on Z-Wave devices, the event table may be used to construct a logical ordering of pairing events. An attack that systematically removes all devices from a controller will be clearly evident in the table as a sequence of remove events. On the other hand, the lack of timestamps in the records will make it difficult to associate the events with other timestamped pieces of evidence.

FortrezZ water valve
pairing event

0x3CC0:	04	04	00	00	01	00	17	01	25	31	32	27	70	85	72	86
0x3CD0:	EF	82	00	00	00	00	00	00	00	00	00	00	18	01	20	80
0x3CE0:	70	85	71	72	86	00	00	00	00	00	00	00	00	00	00	00
0x3CF0:	00	00	19	01	25	72	86	71	22	70	00	00	00	00	00	00
0x3D00:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0x3D10:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Figure 35. Event for FortrezZ Water Valve paired to a Z-Stick2.

3.7 Conclusion

Z-Wave is a wireless IoT system implementation deployed in home and business, which makes it likely that the devices will become candidates for digital forensics investigations. The ZM3102 is a common module found in Z-Wave devices, which contains a ZW0301 MPU. The ZW0301 is an extended Intel 8051 MPU that provides Z-Wave specific communication and application services. The module contains Flash memory and interfaces with an external EEPROM memory; either memory may be

reprogrammed in the field and is vulnerable to a firmware modification attack.

This chapter makes several contributions towards a capability to perform forensic analysis on non-volatile memories of Z-Wave devices. The hardware and software architectures of the Z-Wave ZW0301 are identified. Methods for extracting memory images are provided. Analysis of the memory images identify the source code type and compiler. Several important data structures are discovered in the EEPROM, including a protocol information table, node adjacency table, event table, and controller capability record. The structures may serve as sources of digital artifacts. The tools developed during this research, including *Dump-Zprom* and *readINITSEC*, are made available under the Github AFITWiSec group [AFI15].

Future Work.

The RAM and extended RAM usage of the ZW0301 Z-Wave transceiver should be analyzed to improve efficacy within the forensics domain. This includes reverse engineering the SFR set in RAM to identify critical I/O functions specific to the ZW0301 processor. For example, preliminary analysis of SFR usage suggests the registers involved in SPI transactions. Register 0xD3 provides a clock, known as the SCK pin. The data register used for byte transactions is identified as register 0xD5. The selector pin is in 0xD4. More reverse engineering needs to be performed to identify registers involved with radio transactions and other I/O functions, such as the ADC and triac controller. A volatile memory development and analysis capability is currently underway and the results are being prepared for a future journal publication. The details of the results are summarized in Chapter VII.

The work herein focuses on examining the ZW0301 module for artifacts because of its ubiquity in commercially available Z-Wave devices. However, more artifacts may exist in vendor-specific memories on the device mainboard. For example, the

Vera-E controller has a large NAND Flash memory on the board that likely contains an information base of many Z-Wave devices, configuration settings, and log files of the Z-Wave system. These should be investigated further. In Chapter IV, the NAND Flash of a Vera-E controller is examined to discover the location of a key file containing several security keys used by the Z-Wave security layer.

An open question is if the ZW0301 is capable of executing instructions stored in the external EEPROM. This may provide an additional threat vector. Finally, the ZW0301 may eventually be replaced with the ZW0501. Analysis should be performed on the new module to determine how it differs from what is known about the ZW0301.

IV. The Z-Wave Security Layer and its Susceptibility to Key Extraction and Replay Attack

4.1 Introduction

Z-Wave devices provide a variety of sensing and actuating functions, which may be composed with user-defined rules in a wireless network to realize and perform IoT applications. Traditional applications include climate control, smart-energy, security monitoring, and safety monitoring; however, the user is free to realize alternative applications. As a consequence, humidity values from a thermostat from one vendor may be used to determine the illumination color of a light bulb provided by a different vendor. Devices use the network to report events, send commands, and share knowledge. To facilitate the large set of applications, the application layer is partitioned into classes. Each class contains a well-defined set of commands. The device manufacturer determines which command classes are supported by the device and includes the appropriate application layer functionality in the firmware distributed with the product.

Given the adversary has the ability to transmit arbitrary Z-Wave frames using a SDR, then it is easy for them to control the network. By eavesdropping upon a target Z-Wave network with a SDR, one may learn network identifiers, including the home ID and several node IDs [BFH⁺15]. This information may be used to construct forged Z-Wave frames and send them to targeted nodes in the network. As long as the frame containing the application layer command arrives error free and is supported by the device, the forged command is executed. Users may tolerate this insecurity for some applications, such as domestic lighting; however, this is not the case for more critical applications such as safety and security. For example, a door lock that can be opened arbitrarily provides a false-sense of security. Regarding safety applications,

the user installing a Z-Wave water valve may not appreciate anonymous forged frames disabling their home water supply. To address arbitrary access, Z-Wave provides a mechanism to transmit and receive authenticated and encrypted frames using the *security command class*.

Security is provided as an application layer command class. Its primary function is to provide immutable confidentiality and integrity properties to the application layer using a *secure network frame transaction*. During a transaction, an application layer command is encrypted, signed, and encapsulated into a secure network frame. This frame is transmitted to the recipient, which verifies the signature to determine if the encapsulated application command should be executed. If the frame is authenticated, then the recipient decrypts the encapsulated application command and executes it.

The manufacturer determines which command classes are implemented on their device [Pae13]. A subset of the implemented command classes are designated as requiring security, which is declared by the device when it is paired to a network. For example, a door lock vendor may mandate the use of the security command class for all lock actuation commands. As a result, unlock door commands received outside of a secure frame are ignored. This includes incorrectly signed, encrypted, or replayed frames.

The implementation of the Z-Wave security layer is proprietary. Access to the design and implementation details of the layer requires adhering to a NDA and paying a fee [Sig15]; however, there are two publicly available sources. In [FG13], a significant portion of the implementation details regarding the security layer is exposed; however, aspects outside the scope of their attack, such as nonce generation, are ignored. Alternatively, the source code in [Ope15b] provides a compatible implementation of the security layer for PC-based Z-Wave controllers.

The contributions of this chapter are as follows. First, the models presented

in [FG13] and [Ope15b] are combined and validated through experimentation on a Z-Wave network. Second, the missing pieces of the security layer are realized by reverse engineering device firmware and invoking the security layer over the Z-Wave network using a SDR. Third, the security of the layer is evaluated; specifically, the feasibility of key extraction and replay attack are explored. While several key vulnerabilities are discovered, experimental results provide evidence that replay attack by forcing a nonce collision is currently infeasible.

The remainder of this chapter is as follows. Section 4.2 conveys the devices, procedures, and generalized experiment setup used for the work herein. Section 4.3 provides the synthesized security layer model as a function of [FG13], [Ope15b], reverse engineering, and protocol analysis. Section 4.4 explores key extraction and replay attacks on the devices. The conclusion and future work are given in Section 4.5.

4.2 Methodology

A comprehensive model of the Z-Wave security is derived from protocol analysis, firmware analysis, and tool development for three Z-Wave devices, where each uses the Z-Wave security layer. The devices under study are depicted in Figure 36, including a MiCasa Verde Edge controller, a Yale door lock, and an Aeon Labs Multisensor 4. Having a proprietary ZW0501 transceiver, the controller and multisensor are 5th generation Z-Wave devices. The door lock utilizes a ZW0301 Z-Wave transceiver and is considered a third generation device.

Firmware Analysis.

From [BRMM16], the ZW0301 is a proprietary Intel 8051 core to include a FSK modulator and region dependent RF modulator. The firmware is located in a 32 KB internal flash memory. The three devices under study also utilize a SPI EEPROM



Figure 36. Devices used for the study are the Yale door lock (lower-left), Aeon Multi-sensor4 (upper-left), and MiCasa Verde-Edge controller (right).

for non-volatile data storage, which is at least 16 KB in size. With regard to volatile memory, the ZW0301 contains 256 bytes of RAM and 2 KB External RAM (XRAM). The internal RAM address space is overloaded with specific purpose, including SFRs, stack memory, bit-addressable memory, and several working register sets to allow fast context switching [Ste05]. In [BRMM16], it is shown that the non-volatile memory regions, the internal flash and external SPI EEPROM, of the ZW0301 may be extracted and analyzed using a GoodFET42 [Goo15], an EEPROM programmer, and the serial port on the ZW0301. The Yale door lock uses a ZM3202N package to house the ZW0301, RF filter, and oscillator. Methods of extracting flash memory of the ZW0501 are currently unknown, so only the firmware of the Yale door lock is analyzed. Because the EEPROM is external to the Z-Wave transceiver, the EEPROM for each device may be extracted for analysis.

Firmware analysis is used to discern how intrinsic security operations, such as

nonce generation, are conducted. Using the methods described in [BRMM16], the firmware and external EEPROM data memory are exported to files for analysis. With considerable effort, the firmware is disassembled in IDA for the 8051 architecture and regions pertaining to the security frame transaction are identified. For example, Figures 37 and 38 provide the initial values in the Substitution Box (S-BOX) and inverse tables extracted from the Yale door lock firmware. These tables and the low level mixing operations related to AES are found near the bottom of the firmware image. Additional findings are presented in Section 4.3.

```
00005c0: 7401 93fa 7402 93f9 2263 7c77 7bf2 6b6f
00005d0: c530 0167 2bfe d7ab 76ca 82c9 7dfa 5947
00005e0: f0ad d4a2 af9c a472 c0b7 fd93 2636 3ff7
00005f0: cc34 a5e5 f171 d831 1504 c723 c318 9605
0000600: 9a07 1280 e2eb 27b2 7509 832c 1a1b 6e5a
```

Figure 37. Beginning of the AES S-BOX table embedded in a Yale door lock firmware.

```
0000660: 8846 eeb8 14de 5e0b dbe0 323a 0a49 0624
0000670: 5cc2 d3ac 6291 95e4 79e7 c837 6d8d d54e
0000680: a96c 56f4 ea65 7aae 08ba 7825 2e1c a6b4
0000690: c6e8 dd74 1f4b bd8b 8a70 3eb5 6648 03f6
00006a0: 0e61 3557 b986 c11d 9ee1 f898 1169 d98e
```

Figure 38. Beginning of the AES inverse S-BOX table embedded in a Yale door lock firmware.

Protocol Analysis.

The role of the security layer is to provide confidentiality and integrity properties to frames transmitted over RF between devices. Beyond firmware analysis, it is also useful to observe the security layer messages sent over the air for devices under study. The Z-Wave SDR stack, used in Chapter VI, is deployed for the work herein. The SDR stack provides a passive collection and active participation in the experimental Z-Wave network. Passive collection is achieved by forwarding promiscuously observed

and demodulated frames to a custom Wireshark dissector for Z-Wave, which is extended from the dissector originally developed for [BFH⁺15]. Because the security layer is a protocol, messages are exchanged between multiple devices. The Linux library libZwave is developed in Chapter VI to allow participation in Z-Wave networks and is used herein for the same purpose within the scope of the security layer.

Many of the figures presented in this chapter use a Wireshark dissector output to display protocol transactions. Observed frames are listed in chronological order. The output for each frame provides summary information of its properties; Chapter VI provides a description of these fields. For this work, the dissector output is modified to also include the byte value of the invoked command in parentheses after the command class string.

The authors of [FG13] claim the security properties of confidentiality and integrity are provided to Z-Wave frames through the use of AES cipher operations. Two tools are developed to validate encryption, decryption, and authentication as specified in [FG13]. First, a C++ program *decryptPCAPNG* parses through PCAP Next Generation (PCAPNG) files captured using the SDR stack to identify secure frame transactions. For each transaction, it recovers the Initialization Vector (IV) and applies the provided encryption key to decrypt the frame. It is also able to derive the encryption key if only the network key is provided. The plaintext of each secure frame is printed to the console along with its packet number as assigned by Wireshark to cross-reference with the encrypted frame in the dissector output.

Encryption and authentication are validated through the development of a C++ program *OpenBarley*. Like *decryptPCAPNG*, it also uses libcryptopp for cipher operations; however, the program also uses libZwave to interact with a target device over a SDR. The user provides the tool with a set of keys, home ID, source ID, destination ID, and an application layer command. The tool performs a secure frame transaction

with the destination ID, posing as the source ID. Embedded within the secure frame is the application layer command. If the secure frame is correctly encrypted and signed, then the application layer command is executed on the remote device.

Analysis Process.

Before the claims in [FG13] may be verified, an effort is made to discover where the keys are located in the MiCasa Verde controller and the Yale door lock. The keys are first discovered in the MiCasa Verde filesystem, making it trivial to locate in the Yale door lock. Having the keys, decryption is validated by capturing encrypted frames sent during a door unlock event, having decryptPCAPNG decrypt the frames, and analyzing the decrypted frames to determine if they correlate with the activity. Encryption and authentication are validated by using OpenBarley to send an unlock door command to the door using a secure frame. Given the door unlocks, the frame is correctly signed and encrypted.

Details omitted in [FG13] are also revealed through analysis and experimentation, including the complete derivation of the authentication and encryption keys, the instantiation of the bootstrap key set, and a reverse engineering of the nonce generation mechanism. With a more complete model of the security layer, it is analyzed by inspection method for weaknesses. Methods for extracting keys from the device are explored and two are demonstrated on a Z-Wave network. Based on the firmware analysis results, the nonce generating process is evaluated to determine the feasibility of forcing a nonce collision through brute force.

4.3 The Z-Wave Security Layer

The Z-Wave security layer is presented herein as a functional decomposition of the synthesis of work performed in [FG13], [Ope15b], and additional work. The intent

is to provide the reader with a complete picture of the current reverse engineering model of the security layer. The security layer provides a mechanism to deliver Z-Wave commands with properties of confidentiality and integrity using symmetric key cryptography. While several keys are known by every device, others are shared during device inclusion or remain private to the device.

The Secure Frame Transaction.

Herein, a *secure frame* is defined as a signed and encrypted frame, containing an embedded application layer command, sent over the Z-Wave network from a sender to a recipient. At a high level, the sender encrypts a payload using an encryption key. The payload, along with other fields in the frame, are signed using an authentication key and a signature is appended to the secure network frame. After these activities, the secure frame is sent to the recipient. Also in possession of the same authentication key, the recipient generates a new signature using the information included in the frame and compares it with the signature included in the frame. If valid, the recipient considers the secure frame to be originated from a trusted source and assumes its contents have maintained integrity while in transit. The recipient decrypts the validated payload and allows the application layer handler to process the message as a normal command. To prevent replay of previously observed secure network frames, each transaction generates a new IV, which is established cooperatively using each device's nonce round key before any cryptographic operations are applied to the target payload. A jointly derived IV is used at both the secure frame sender and recipient for cipher-based activities regarding confidentiality and integrity of the transaction.

Three types of messages are used during a secure frame transaction. The first message is initiated by the sender, which requests that the recipient provide a portion

of the transaction IV, known as a *nonce*, using a *Get Nonce* message [FG13]. The recipient provides this information in a *Nonce Report* message and is the second type of message. With the IV established, the sender is able to create the third message type, the secure network frame, to include the encrypted and signed payload. The secure frame includes the encrypted application layer command, a signature, and the portion of the IV generated by the sender. To send a new secure frame, a new transaction is initiated; however, large frames may be fragmented into multiple secure frames within a single transaction.

Two complete secure frame transactions are observed in Figure 39 as a result of an SDR packet capture of Z-Wave frames exchanged between a MiCasa Verde controller and Yale door lock. The first transaction occurs in frames 1-1019 while the second is shown as frames 1021-1029. Both transactions involve the controller sending a secure frame to the door lock. Regarding the first transaction, the controller makes three attempts to request a nonce from the door lock in frames 3, 5, and 1009 using a nonce get message. Because the door lock is battery powered, the controller must send it a wakeup pattern of short frames, known as a beams, to temporarily bring the device to an active state. The beam frames are omitted from the figure for clarity. Once the door lock is awake, it responds with a nonce report message at frame 1013. The secure frame is transmitted from the controller to the door lock in frame 1017. Regarding the second transaction, the nonce get, nonce report, and secure frame are shown as frames 1021, 1025, and 1027, respectively.

Establishing the Transaction IV.

The 16-byte IV used for cryptographic operations during a transaction is composed of two 8-byte nonces. The sender nonce occupies the upper 8-bytes of the IV and the recipient's nonce is located in the lower 8-bytes [FG13]. Sender and recipient IV

No.	Time	Protocol	Info
3	0.040...	Zwave	MAC: Singlecast(11) [0xe56194a9 1->25] APP: Security(0x40)
5	0.079...	Zwave	MAC: Singlecast(11) [0xe56194a9 1->25] APP: Security(0x40)
1009	1.242...	Zwave	MAC: Singlecast(12) [0xe56194a9 1->25] APP: Security(0x40)
1011	1.249...	Zwave	MAC: ACK(12) [0xe56194a9 25->1]
1013	1.257...	Zwave	MAC: Singlecast(1) [0xe56194a9 25->1] APP: Security(0x80)
1015	1.264...	Zwave	MAC: ACK(1) [0xe56194a9 1->25]
1017	1.341...	Zwave	MAC: Singlecast(13) [0xe56194a9 1->25] APP: Security(0x81)
1019	1.346...	Zwave	MAC: ACK(13) [0xe56194a9 25->1]
1021	11.44...	Zwave	MAC: Singlecast(14) [0xe56194a9 1->25] APP: Security(0x40)
1023	11.44...	Zwave	MAC: ACK(14) [0xe56194a9 25->1]
1025	11.45...	Zwave	MAC: Singlecast(1) [0xe56194a9 25->1] APP: Security(0x80)
1027	11.58...	Zwave	MAC: Singlecast(15) [0xe56194a9 1->25] APP: Security(0x81)
1029	11.58...	Zwave	MAC: ACK(15) [0xe56194a9 25->1]

Figure 39. Two secure frame transactions between a door lock and controller are captured with a SDR.

coherency is achieved at different points of the transaction. The sender is able to realize the transaction IV after it has generated its nonce and received a reply to its request for a nonce from the recipient. The recipient achieves IV coherency upon receipt of the secure frame.

The nonce generating algorithm is discovered in the disassembly analysis of the firmware for a Yale door lock and is provided in Figure 40. A device generates a nonce upon receipt of a *Nonce Get* message or for an outgoing secure frame. A new nonce is generated by encrypting a 16-byte block containing a repeating byte value of 0x5C with the nonce round key in AES Electronic Code Book (ECB) mode. Only the first 8-bytes of the output block is used for the nonce, while the other bytes remain hidden. After a nonce is generated, the nonce key is changed. The new key is derived from the output block of encrypting an input block of a pattern of 0x36 byte values with the current nonce round key in ECB mode. Firmware analysis of the Yale door lock reveals the recipient caching the nonce in the first eight bytes of one of the eight available 16-byte entries before sending it in a nonce report message.

Upon arrival of the secure frame, the recipient must be able to retrieve its 8-byte nonce associated with the transaction. The secure frame contains a *nonce ID* [FG13] single byte field, which is the first byte of the recipient's nonce. The cache is examined for a match, and if found, the 8-byte nonce included in the secure frame is copied to

```

; Input:
; PTR for where NONCE will go
;
; Encrypts 0x5Cs using ECB with KEY at XRAM_703
; Copies first 8 bytes
; Generates a new key for next time by encrypting
; 0x36s with KEY at XRAM_703
; OUTBLOCK becomes new key (memcpy)

Generate_NONCE_round:
1 mov     DPTR, #0x6CE
2 lcall   PTR_STORE_XRAM_R3_R2_R1_atDPTR
3 mov     R3, #1
4 mov     R2, #0
5 mov     R1, #0x39 ; '9'
6 mov     R5, #0x10
7 mov     R7, #0x5C ; '\' ; next instruction calls memset,
; so this is saying
; at XRAM_0x039, set value 0x5C for 16 bytes
8 acall   ECB_ENCRYPT_R7s_block_with_key_at_XRAM_703 ; sets 0x60D ptr to XRAM_029
9 mov     DPTR, #0x6CE
10 lcall  PTR_LOAD_XRAM_atDPTR_to_R3_R2_R1
11 mov     R7, #8 ; The first 8 bytes of the OUTBLOCK is copied
; to PTR at XRAM_6CE
12 acall   memcpy ; (to=PTR(R3R2R1), from=XRAM_60D, len=R7);
13 acall   ECB_ENCRYPT_0x36sBlock_with_key_at_XRAM_703 ; specialized Encrypt ECB operator
; INBLOCK is init'd to all 0x36's
; KEY is XRAM_703
; OUTBLOCK (i.e., XRAM_029)
; is set as memcpy FROM/SRC ptr before returning
14 mov     R2, #7
15 mov     R1, #3
16 mov     R7, #0x10 ; OUTBLOCK is copied to the location holding the key
; (Key is updated)
17 ajmp    memcpy ; (to=PTR(R3R2R1), from=XRAM_60D, len=R7);
; End of function Generate_NONCE_round

```

Figure 40. The nonce generating algorithm found in a Yale door lock firmware image.

the last half of the cache entry. Thus, the transaction IV is realized by the recipient for the given cache entry. The firmware analysis also reveals additional iterating through cache entries to invalidate duplicate entries of the provided nonce ID by writing a 0x00 to the first byte. While untested, this scheme may permit up to eight concurrent secure frame transactions with other devices. Interestingly, cache entries are agnostic to the transaction sender, implying the nonce is first come, first served. Further study is required to determine if a race condition exists where another device uses the nonce in a secure frame, sent to the recipient, before the transaction sender.

The Secure Network Frame.

The secure frame is an application layer Z-Wave frame. At the RF layer, the frame includes a PPDU header, MPDU header, optional network layer, and application payload. The first two bytes of the application layer payload determine the semantics of the remaining bytes in the frame, where the first byte designates the *command class* and the second byte determines the command to be executed within the class. For a secure network frame, these bytes are 0x98 and 0x81, respectively. Application layer commands requiring confidentiality and integrity are embedded into the secure frame.

The application layer security layer, shown in Figure 41, contains the fields necessary to decrypt and authenticate the encrypted payload. The upper IV field contains the 8-byte nonce generated by the source node, allowing the recipient to realize the transaction IV. The upper IV is followed by the encrypted payload, nonce ID of the recipient's nonce, and an 8-byte Cipher Message Authentication Code (CMAC) signature.

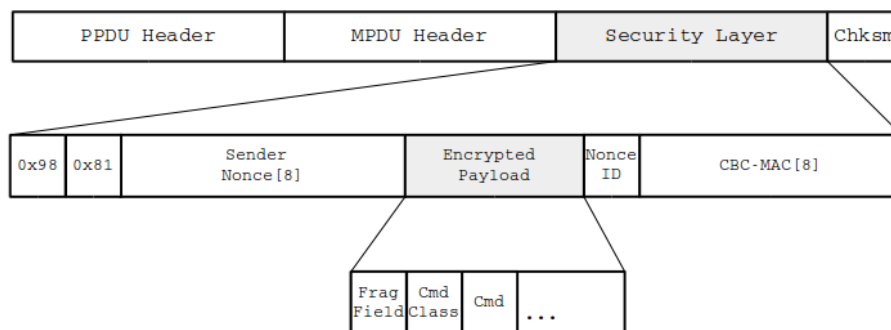


Figure 41. The application layer fields of a secure frame in the context of a Z-Wave PPDU.

The encrypted payload contains a single byte fragmentation field followed by the embedded application layer message [Ope15b]. The field allows the recipient to collect the fragmented blocks before applying authentication and decryption on the complete

payload. If this field is zero, fragmentation is not used. At a minimum, the application layer message is composed of two 1-byte fields, which are the *command class* and *command* respectively. Additional parameters may be appended to the minimum size message as a function of the particular command class and command.

The encrypted payload length varies between secure frames; however, a length field is not included. The payload length is calculated after discounting the fixed fields from the MPDU header length field. The fixed fields include a 9-byte MPDU header, 2-byte application header, 8-byte sender nonce, 1-byte nonce ID, 8-byte signature, and 1-byte checksum.

Secure Frame Authentication.

The secure frame is authenticated with a CMAC signature derived from the process shown in Figure 42. It is generated by encrypting a message derived from a *subset* of the fields in the secure frame using an AES cipher in Cipher Block Chaining (CBC) mode using a block size of 16 bytes [FG13]. Only the lower 8-bytes of the cipher text block are used as the signature [FG13].

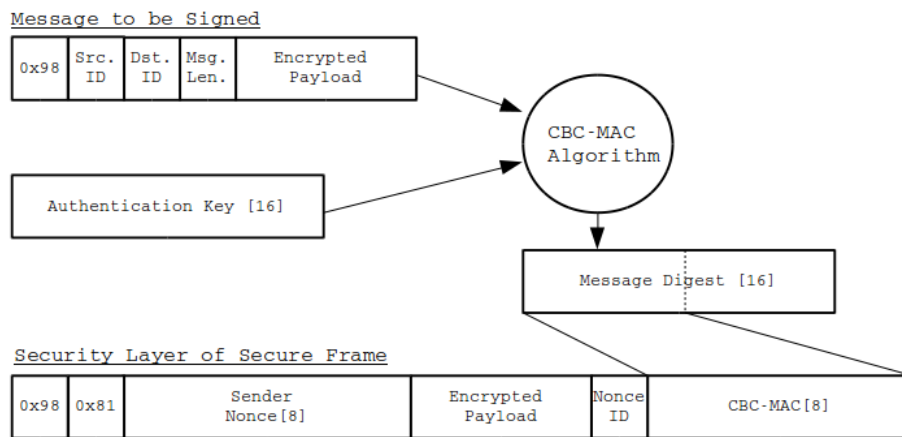


Figure 42. The CMAC derivation process.

Figure 42 provides the field ordering of the message used to generate the CMAC. The first four bytes of this message are always the security command class ID, source

ID, destination ID, and length of the message [FG13]. The remaining portion of the message is the encrypted payload. The scope of data integrity is limited to the fields included in the signed message. Several fields are omitted, including the sequence number, frame type, checksum, and the entire routing layer. This choice may have been for performance reasons; however, Chapter VI identifies several routing layer vulnerabilities that could be addressed by providing integrity to the routing layer.

While the inclusion of the source ID in the message used to generate the signature may appear to allow node ID level authentication, every device uses the same key. The recipient may only determine the message is signed with the shared key and the source ID is not changed while in transit. A node in possession of the authentication key may present any node ID as the source ID of a secure frame, and the recipient is unable to detect the forgery. Implicitly, the system assumes nodes in possession of the authentication key are invariantly trusted.

Sending a Secure Network Frame.

A given application layer message requiring confidentiality and integrity properties is prepared as follows. After the sender performs the nonce exchange steps required to realize the transaction IV, it copies its generated nonce into the first 8 bytes of the security layer so the recipient is also able to realize the transaction IV. The encrypted payload is prepared by prefixing the application layer message with the fragmentation byte. The message is encrypted using the encryption key in Output Feedback (OFB) mode [FG13], [Ope15b] and placed into the secure frame. The first byte of the recipient's nonce is placed in the nonce ID field. The CMAC message is constructed and signed using the authentication key in CBC mode [FG13] and placed at the end of the secure frame, which is passed to the transmission layer to send the frame [Ope15b].

Receiving a Secure Network Frame.

Upon receipt of the secure frame, the recipient uses the nonce ID field to locate the transaction IV cache entry. Once found, the the upper IV field is transferred to the upper portion of the cache entry. Next, the recipient authenticates the frame by composing the CMAC message using the same subset of fields used to generate the original CMAC [FG13]. Like the sender, a 16-byte signature is generated by encrypting the CMAC message in CBC mode using the authentication key and transaction IV. The first eight bytes of the signature are compared with the CMAC field of the frame. If the bytes differ, then the frame is considered unauthenticated and is silently discarded. Otherwise, the frame is considered authenticated and moves to decryption. Decryption is performed on the encrypted payload using the encryption key and transaction IV in OFB mode. The resulting plaintext is passed to the application layer handler for processing.

Key Management.

The Z-Wave protocol provides confidentiality and integrity services for frames sent over the network by the utilization of cryptography, where security is preserved by keeping several symmetric keys hidden from the adversary. Confidentiality is maintained using an encryption key. Frame integrity and weak authentication are preserved using an authentication key. Both keys are derived from a network key, shared by the controller when the device is added to the network. Because these keys are used for security operations, they are designated herein as the *operational key set*. Every device in a given Z-Wave network shares the same operational key set. Every device uses an identical *bootstrap key set*. Like the operational key set, it contains a network, authentication, and encryption key and is used during the inclusion process to covertly send the network key in a secure frame transaction. Finally, a nonce

round key is an internal key maintained by each device used to generate IVs for cryptographic operations to reduce replay attack effectiveness. The derivation and memory location of each key is described in the following subsections.

Bootstrap Key Set.

Controllers use a *bootstrap key set* to send a signed and encrypted network key to new devices when they are added to their network. Using a network key of all zeros [FG13], the bootstrap encryption and authentication keys are derived using the operations described in Section 4.3 and are shown in Figure 43. Once a device is joined to a network, it uses the operational key set in lieu of the bootstrap key set.

```

Network Key:  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Encryption Key: 85 22 71 7d 3a d1 fb fe af a1 ce aa fd f5 65 65
Authentication Key: 9a da e0 54 f6 3d fa ff 5e a1 8e 45 ed f6 ea 6f

```

Figure 43. The bootstrap keys are a set of keys, common to all Z-Wave devices, used to send the controller’s network key to a device during network inclusion.

Network Key.

A device is added to the network using an inclusion operation, where the controller and device exchange information on application and network capabilities. Given the controller and device support a compatible security layer, they share security information. The device declares a subset of its supported command classes requiring confidentiality and integrity services and the controller provides the device with a network key in the manner described in [FG13]. To validate their claim, a Yale door lock is paired to a MiCasa Verde Edge controller while an SDR is capturing Z-Wave frames. The results are shown in Figure 44.

Three security command class messages are used during the exchange, which are commands 4, 5, and 6. From the OpenZwave implementation of the Security command class, these commands are *Scheme Get*, *Scheme Report*, and *Network Key Set*,

No.	Time	Protocol	Info
14684	1844.59...	Zwave	MAC: Singlecast(8) [0xe56194a9 1->25] APP: Security(0x04)
14685	1844.60...	Zwave	MAC: ACK(8) [0xe56194a9 25->1]
14686	1844.61...	Zwave	MAC: Singlecast(3) [0xe56194a9 25->1] APP: Security(0x05)
14687	1844.62...	Zwave	MAC: ACK(3) [0xe56194a9 1->25]
14688	1844.69...	Zwave	MAC: Singlecast(9) [0xe56194a9 1->25] APP: Security(0x40)
14689	1844.70...	Zwave	MAC: Singlecast(4) [0xe56194a9 25->1] APP: Security(0x80)
14690	1844.71...	Zwave	MAC: ACK(4) [0xe56194a9 1->25]
14691	1844.85...	Zwave	MAC: Singlecast(10) [0xe56194a9 1->25] APP: Security(0x81)
14692	1844.86...	Zwave	MAC: ACK(10) [0xe56194a9 25->1]

Figure 44. Packet capture of a Vera Edge Controller providing the network key to a Yale door lock with a node ID of 25 using a secure frame transaction.

respectively [Ope15b]. Figure 44 captures the key exchange protocol, where the controller is node ID 1 and the door lock has node ID 25. Frame 14684 shows a Scheme Get message sent by the controller to the door lock. The door lock sends its security capabilities to the controller in the Scheme Report message, shown as frame 14686. Following this frame is a secure frame transaction, where frame 14691 is the secure frame sent from the controller to the door lock.

Using the bootstrap key set, decryptPCAPNG is used to decrypt the secure frame in the packet capture. The results are shown in Figure 45. The first byte of the payload is the fragmentation byte, which is zero. The second and third bytes reveal the message is a security command class network key set message. The 16-byte network key begins at the fourth byte.

```

PacketNum=14691 is an encapsulated encrypted frame
nonceId=0xf9 is FOUND in the cache.
Encrypted Payload:  20 4c d1 62 c7 a8 3e 6b 60 a9 12 2e 89 af 16 35 fe 2f 19
Decrypted Payload:  0 98 6 e9 7a 56 31 cb 56 86 fa 24 45 e ba 10 3f 94 5c

```

Figure 45. Frame 14691 is decrypted using decryptPCAPNG and the bootstrap key set to reveal a security set key message containing the network key. The network key begins at the fourth byte of the decrypted payload.

The network key is stored in non-volatile memory so rekeying is unnecessary after a loss of power or reset. Given a loss of volatile memory, firmware analysis shows the network key is loaded into volatile memory and the encryption and authentication keys are recovered using the key derivation algorithm. The location of the key is manufacturer specific and depends on the available hardware of the device. To

illustrate this aspect, an attempt is made to find the non-volatile memory location of the network key for the Yale door lock and MiCasa Verde controller.

The MiCasa Verde controller uses an embedded Linux Operating System (OS) and stores its *operational key set* in a 48 byte file in its filesystem located at `/etc/cmh/keys`. The file may be extracted by using its web interface to download a compressed archive of the controller configuration or using the provided secure shell interface. Figure 46 shows the byte values of the key file extracted from the controller. The first 16-bytes of the file is the network key. The authentication and encryption keys follow. The controller also provides a EEPROM to the ZW0301 chip using a SPI bus, which is used to stored several data structures described in [BRMM16]. The operational key set is not observed in the controller EEPROM.

```
$cd /etc/cmh
$ls
alerts.json      HW_Key           user_data.json.lzo.1
cmh.conf         HW_Key2          user_data.json.lzo.2
devices          keys             user_data.json.lzo.3
dongle.3.83.dump.0 last_report      user_data.json.lzo.4
dongle.3.83.dump.1 PK_AccessPoint  user_data.json.lzo.5
dongle.3.83.dump.2 servers.conf.default
dongle.3.83.dump.3 sync_kit         vera_model
dongle.3.83.dump.4 sync_rediscover  wan_failover
ergy key         user_data.json.luup.lzo
first_boot       user_data.json.lzo
$xxd ./keys
00000000: e97a 5631 cb56 86fa 2445 0eba 103f 945c  .zV1.V..$E...?.\
00000010: 620d 486c 6a65 2122 afe1 086c 79e6 3740  b.Hlje!"...ly.7@
00000020: eec9 ef96 a155 a3d3 02a1 8441 f5f3 7ea0  .....U.....A..~.
$
```

Figure 46. Binary dump of the operational key set file found in MiCasa Verde Edge controller, shown in the context of its filesystem directory.

Like the controller, the Yale doorlock has an EEPROM chip for storing non-volatile data structures. In this case, the network key observed in the MiCasa Verde key file is found a memory address 0x012D, shown in Figure 47. The network keys found in the MiCasa Verde controller, the Yale door lock, and the key exchanged during pairing are consistent. The activity confirms decryptPCAPNG correctly de-

crypts the payload for secure frames. Since decryptPCAPNG implements Z-Wave decryption according to [FG13] [Ope15b], their claims regarding decryption are also verified.

```

0000100: 0000 0000 0000 0000 0000 0000 0000 0000 .....
0000110: 0000 0000 0000 0000 0000 0000 0000 0000 .....
0000120: 0000 0000 0042 0100 0000 0000 81e9 7a56 .....B.....zV
0000130: 31cb 5686 fa24 450e ba10 3f94 5c56 0000 1.V..$E...?.\V..
0000140: 0000 0000 0000 0000 0000 0000 0000 0000 .....
0000150: 0000 0000 0000 0000 0000 0000 0000 0000 .....

```

Figure 47. Segment of the EEPROM of a Yale door lock containing the network key at address 0x012D.

Encryption and Authentication Keys.

The encryption and authentication keys are derived using an identical process, where a 16-byte input block is encrypted in AES ECB mode using the network Key [FG13].

While Fouladi’s paper provides the relationship between the keys, it does not reveal the input block values required to derive the keys. Given one has access to the operation key set, the input block values may be discovered by reversing the process. Decrypting an input block containing the encryption key using the network key in AES ECB mode reveals its input block pattern; the same is true for the authentication key. The aforementioned process is performed on a set of keys extracted from a MiCasa Verde controller using a custom software tool based on the gcrypt C library. The console output is provided in Figure 48 and labels the input block for the authentication key as *A_K*. The resulting output block is labeled *A_Seed*. The encryption input and output blocks of the decryption operation are provided in the last two lines of the figure. The figure shows that the derivation for each key uses a different input block containing a repeating pattern of values. The encryption key is derived with an input block containing all 0xAA’s, and the

authentication key uses an input pattern of all 0x55's.

```
./getPwds ~/Downloads/etc/cmh/keys
gcry_cipher_open worked
gcry_cipher_setkey worked
gcry_cipher_decrypt worked
A_K: : 62 d 48 6c 6a 65 21 22 af e1 8 6c 79 e6 37 40
A_Seed: : 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55
gcry_cipher_decrypt worked
E_K: : ee c9 ef 96 a1 55 a3 d3 2 a1 84 41 f5 f3 7e a0
E_Seed: : aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa
```

Figure 48. Reversing the key derivation process to derive the ECB inblocks required to generate the encryption and authentication keys from the network key.

By analyzing the firmware, it is determined that the input blocks are set using a *stdlib::memset* function, which writes a provided byte value a given number of times from a starting memory address inside a function call at code address 0x30AE. Figure 49 shows the 8051 disassembly of the key derivation algorithm, where the authentication key is derived first, followed by the encryption key. Function *code_30AE* calls *memset* on the input block 16-byte buffer and sets each byte to the value in register *R7*. The figure shows that *R7* is loaded with an immediate value of 0x55 on line 1 and line 7 is set to 0xAA. The immediate load of the address immediate values are derived during compilation of the firmware image.

Figure 49 also reveals the location of the operational key set in volatile memory. The *ENCRYPT_mode_ECB* function takes three pointers: the input block, output block, and the key. On lines 3-5, the output block is set to XRAM address 0x01AC. The output block is set to XRAM 0x01BE on lines 9-11. The disassembly reveals the addresses of the keys are constant values, determined at compile time of the firmware image. The scope of variation of these addresses for different firmware images is left as future work.

Figure 49 also reveals the memory location of the network key. The third parameter of the *ENCRYPT_mode_ECB* function is a pointer to the key. Since the encryption

```

DeriveEnc_and_Auth_keys:
1 mov     R7, #0x55 ; 'U'
2 acall    code_30AE ; Sets XRAM_16 to 16 bytes of R7
                  ; Places ptr to XRAM_16 at XRAM_57F
3 mov     R2, #1
4 mov     R1, #0xAC ; 'X' ; Ref to address 0x1AC
5 acall    code_3089 ; Stores current ptr at XRAM_582
                  ; sets R2 and R1 to 0x00 and 0xFA respectively
6 lcall    ENCRYPT_mode_ECB ; (INBLOCK=XRAM_582, OUTBLOCK=XRAM_57F, KEY=PTR(R3,R2,R1))
7 mov     R7, #0xAA ; 'A'
8 acall    code_30AE ; Sets XRAM_16 to 16 bytes of R7
                  ; Places ptr to XRAM_16 at XRAM_57F
9 mov     R2, #1
10 mov    R1, #0xBE ; '+'
11 acall    code_3089 ; Stores current ptr at XRAM_582
                  ; sets R2 and R1 to 0x00 and 0xFA respectively
12 ljmp     ENCRYPT_mode_ECB ; (INBLOCK=XRAM_582, OUTBLOCK=XRAM_57F, KEY=PTR(R3,R2,R1))
                  ; End of function DeriveEnc_and_Auth_keys

```

Figure 49. Intel 8051 disassembly of the key derivation algorithm, located in the firmware of a Yale door lock.

and authentication keys are derived from the network key, the third parameter points to the memory location of the network key. For the analyzed firmware image, the memory location is at XRAM 0x0FA because a pointer to XRAM is loaded in a set of working registers on lines 5 and 11.

Nonce Round Key.

The nonce round key is used to generate a low entropy IV for each secure network frame transaction by exploiting the security properties of the AES cipher. Unlike the operational and bootstrap key set, a device nonce round key is private. From Section 4.3, a new nonce round key is generated by encrypting an input block of repeating byte values of 0x36 with the current nonce round key in AES ECB mode. Since a nonce round key is a function of its predecessor, the initial nonce round key is generated in a different manner.

Firmware analysis of the disassembly regarding the nonce key initialization reveals the initialization algorithm. The initial round key is created in a series of rounds, where the number of rounds is provided as a function parameter. In each round,

a 16-byte block is encrypted in ECB mode using a randomly generated key. An Exclusive OR logical operation (XOR) operation is performed on the input and output blocks of the cipher operation to derive the input block of the next round. For the first round, the input block is set to a repeating pattern of 0xA5 byte values. After the required number of rounds are completed, the input block becomes the initial nonce round key. The next round key is immediately calculated by encrypting an input block of a pattern of 0x36 byte values with the initial round key. This means the initial nonce round key is never used to generate a nonce.

The random keys used in the generation of the initial nonce round key are generated using a provided hardware Random Number Generator (RNG). From the firmware of a Yale door lock, a string of random bits is generated in a loop, where the *i*th random bit is the state of bit 6 of SFR 0xCA. Further details about the RNG source are not available. Examples of low-cost hardware random sources include floating pins [Ste15], input latches [BKPK16] and the low bits of an ADC [KHL⁺16].

Like the operational key set, the nonce round key is also stored in XRAM at a static memory location. The disassembly provided in Figure 40, at line 8, reveals the nonce round key is stored at XRAM 0x703 of the analyzed door lock firmware. Like the other keys in memory, the value is either set explicitly in the source code or determined during compilation.

4.4 Attack Vectors

Given the existing model of the security layer, two attack vectors are considered. Since Z-Wave relies on cryptography for maintaining integrity and confidentiality properties, acquisition of the operational key set for a given network effectively eliminates security for the devices in the network. Therefore, several methods are explored regarding techniques for an adversary to attain these keys.

Assuming the adversary is unable to learn the operational key set, then an alternative attack vector is to collect and reuse observed frames sent by legitimate devices. Having a collection of legitimate secure frames, the adversary may establish a repertoire of reusable commands. Cryptographic systems utilize counters or *nonce* values to make ciphertexts distinguishable [FS03]. Z-Wave utilizes this aspect to detect and prevent replay; however, its effectiveness has yet to be measured. An attack vector is considered, where a nonce collision is forced through repeated nonce requests to progress the nonce to match the one used for a previously captured frame so it may be replayed.

Operational Key Set Extraction.

With respect to the operational key set, one might consider a scenario where the adversary is able to acquire only one of the keys. The event where the network key is acquired is the most significant, because both the encryption and authentication keys may be derived. If the adversary obtains the encryption key instead, they may decrypt frames. While they are also able to generate secure frames with a properly encrypted payload, they are unable to correctly sign the frames. As a result, they lack the ability to command and control devices through the security layer. The case with the least privilege is when the adversary has only the authentication key, where they are only able to sign and verify signatures of secure frames. Without the encryption key, they are unable to decrypt observed frames. Moreover, they are unable to properly encrypt the secure frame payload. While any transmitted secure frames may be correctly signed, the recipient decrypts the payload after verifying the signature. Having not properly encrypted the payload, the adversary's payload is corrupted by the decryption operation to an undefined state. The exercise reveals the network key provides the most privileges, so it is the primary target of the attack

vector.

Firmware Exploitation to Invoke XRAM Key Leakage.

Each key of the operational key set is stored in XRAM along with other data items used in Z-Wave operations. Using firmware analysis, a vulnerability may be discovered, allowing one or more of the operational key set to be exposed using an exporting mechanism such as an RF frame or serial transaction. The likelihood of the attack depends on the ability to remotely influence pointers used in IO activities so they point to key data. Such an attack has yet to be discovered on any recovered firmware images; however, the firmware is compiled C code [BRMM16]. From [SPWS13], memory corruption and data leakage is a pervasive issue for non type-safe languages, such as C.

Acquire Network Key during pairing.

A SDR may collect frames during pairing operation and use the bootstrap keys, as done in Figure 45 to learn the network key. With the network key, the adversary is able to derive the operational key set and send arbitrary commands within secure frames. The attack has a limited window of opportunity because the exchange happens only once per included device. However, the attack vector is still reasonable if the target network is persistently monitored by a Z-Wave frame capturing device. Moreover, the adversary may find ways to force the network into inclusion using social engineering. With physical access, a device could be reset or swapped with a new device with the same model number. If the controller acts as an IP gateway and is vulnerable to the rogue node injection technique, a device under the control of the adversary may be added to the network. Having been paired to the network, the device under the control of the adversary holds the network key [FR15] in its memory. [FG13] is first

to declare this a vulnerability; however, they disregard its significance because of the serendipity needed to acquire the network key.

Acquire Network Key from EEPROM.

Since Z-Wave uses symmetric key cryptography, the operational key set is the same for all devices in the network. The keys may be extracted from devices with low physical security and used to compromise physically secure devices. To illustrate the issue, one may assume a homeowner installs a Z-Wave door lock for the front door, which uses the secure frame transaction to remotely lock and unlock the door. The door lock stores the network key on the EEPROM, which is located on the interior side of the unit. Access to this memory requires physical access to the interior of the home or physically compromising the door. In other words, given one has access to the interior of the door, acquisition of the network key is no longer necessary.

Enjoying the benefits of IoT, the homeowner extends their Z-Wave network by adding a PIR sensor and smart bulb. They are installed on the front porch and configured together to illuminate the door when motion is detected. By default, the PIR sensor sends out alarm messages using secure frame transactions. Like the door lock, the same network key is stored in its memory; however, the physical security of the PIR sensor is much lower because it is installed outside the home. An intruder only needs to compromise the plastic chassis to gain access to the EEPROM. To demonstrate this vulnerability, an Aeon Multisensor with a PIR motion detection sensor is added to the Z-Wave network containing the MiCasa Verde Edge controller and Yale door lock used in previous examples. Figure 50 shows the PIR sensor contains a ZW0501 Z-Wave transceiver chip in a ZM5101A package. An SPI EEPROM is located in proximity to the chip and is outlined in the figure. The non-volatile memory component is removed from its board and its contents are extracted using

an EEPROM programmer board.

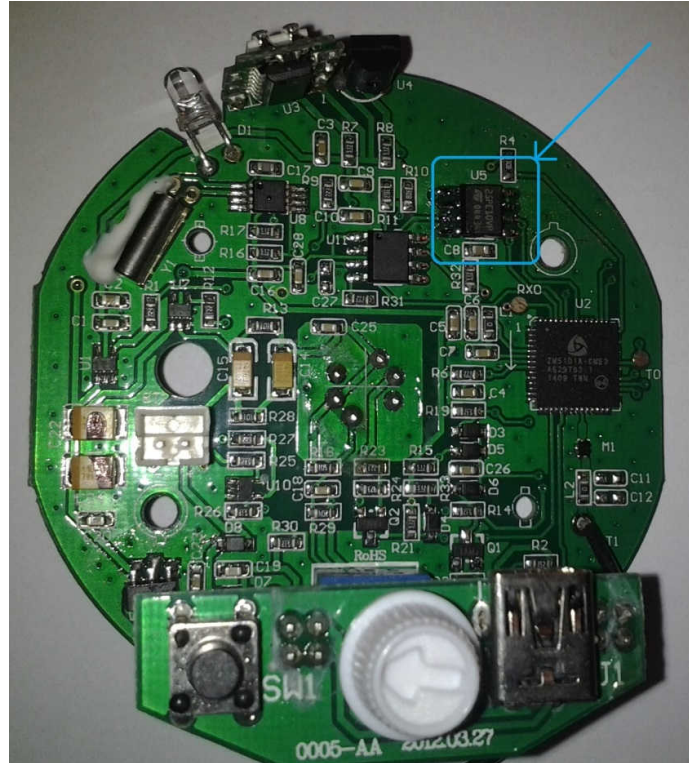


Figure 50. Internal hardware components for the Aeon Multisensor 4, where the EEPROM used by the ZW0501 is highlighted.

Figure 51 shows the network key is in the PIR sensor EEPROM memory, which matches the network key found in the controller filesystem and door lock EEPROM discovered in Section 4.3. The EEPROM memory is largely vacant, making it trivial to locate a varying sequence of 16-bytes within the address space. The key is found at address 0x60A0.

```

0006070: 0000 0000 0000 0000 0000 4200 0000 0000 .....B.....
0006080: 0000 0000 0000 0000 0000 0000 0000 4200 .....B.
0006090: 0000 0000 0000 0000 0000 0000 ff00 0001 .....
00060a0: e97a 5631 cb56 86fa 2445 0eba 103f 945c .zV1.V..$E...?.\
00060b0: 5600 1498 eff1 7275 13cc 4201 0000 0000 V....ru..B.....
00060c0: 4232 6402 a801 0000 0000 0000 0000 0000 B2d.....
00060d0: 0000 0000 0000 0000 0000 0000 0000 0000 .....

```

Figure 51. The network key is located at address 0x60A0 of an Aeon Multisensor 4 EEPROM.

After acquiring the network key from the Aeon Multisensor, an intruder is able

to recover the encryption and authentication keys using the algorithm described in Section 4.3. With these keys, they can send a signed and encrypted unlock door command using a secure frame to attain access into the home.

The vulnerability is verified experimentally using OpenBarley, which implements the encryption and signature preparation mechanisms described in [FG13]. From [Ope15b], the door lock command class is 0x62, where commands 0x01, 0x02, and 0x03 are lock state SET, GET, and REPORT commands, respectively. The door lock is unlocked and locked when its state is 0x00 and 0xFF, respectively. The door lock state is requested before and after OpenBarley sends an unlock command to confirm the unlock command is obeyed. Frames are collected during the activity and decrypted with decryptPCAPNG.

Figure 52 provides a summary of the secure frame transactions occurring between OpenBarley posing as node ID 100 and the door lock, which is now node ID 30. Beam frames, collisions, retransmissions, and ACK frames are omitted for clarity. The figure shows five distinct secure frame transactions. Each request for the door lock state requires two secure frame transactions, one to request the state and one to report it. Secure frames are sent at frame numbers 5, 11, 17, 23, and 29. Figure 53 shows the contents of each secure frame. Frame 5 is a request from OpenBarley to the door lock for its state. In frame 7, the door lock declares its state is 0xFF, which is locked. The door unlock command is set by OpenBarley in frame 11, where it tells the door to SET its state to 0x00, which is unlocked. At this point, the motor actuates on the lock and the bolt is disengaged. In frame 23, OpenBarley again requests the door lock state and the reply is provided in frame 29. The last frame shows the lock state is now open instead of closed. Thus, using the network key extracted from the Aeon Multisensor allows OpenBarley to issue commands to the door lock. The source code for OpenBarley is publicly available and embedded in the electronic distribution of

[BR16].

No.	Time	Protocol	Info
1	0.000000000	Zwave	MAC: Singlecast(0) [0xe56194a9 100->30] APP: Security(0x40)
2	0.012726111	Zwave	MAC: Singlecast(1) [0xe56194a9 30->100] APP: Security(0x80)
3	1.185437329	Zwave	MAC: Singlecast(2) [0xe56194a9 100->30] APP: Security(0x81)
4	1.233160585	Zwave	MAC: Singlecast(2) [0xe56194a9 30->100] APP: Security(0x40)
5	1.364172115	Zwave	MAC: Singlecast(3) [0xe56194a9 100->30] APP: Security(0x80)
6	1.397476991	Zwave	MAC: Singlecast(3) [0xe56194a9 30->100] APP: Security(0x81)
7	13.628035775	Zwave	MAC: Singlecast(0) [0xe56194a9 100->30] APP: Security(0x40)
8	13.796708822	Zwave	MAC: Singlecast(1) [0xe56194a9 30->100] APP: Security(0x80)
9	14.969481241	Zwave	MAC: Singlecast(2) [0xe56194a9 100->30] APP: Security(0x81)
10	54.267378635	Zwave	MAC: Singlecast(0) [0xe56194a9 100->30] APP: Security(0x40)
11	54.367089517	Zwave	MAC: Singlecast(1) [0xe56194a9 30->100] APP: Security(0x80)
12	55.539971027	Zwave	MAC: Singlecast(2) [0xe56194a9 100->30] APP: Security(0x81)
13	55.588388497	Zwave	MAC: Singlecast(2) [0xe56194a9 30->100] APP: Security(0x40)
14	55.609496287	Zwave	MAC: Singlecast(3) [0xe56194a9 100->30] APP: Security(0x80)
15	55.680546315	Zwave	MAC: Singlecast(3) [0xe56194a9 30->100] APP: Security(0x81)

Figure 52. Frames captured while OpenBarley opens a Yale door lock using a key extracted from another Z-Wave device.

The network key resides in non-volatile memory until the device is formally excluded from the network. To demonstrate, the Aeon Multisensor is formally excluded from the network managed by the MiCasa Verde Edge controller. Previous analysis of the Multisensor EEPROM reveals the key is stored at address 0x06A0. After exclusion, the contents of the EEPROM are extracted and Figure 54 shows the memory region, formerly occupied by the network key, is zeroed.

A user may eventually decide to sell a Z-Wave device previously deployed in their network. If an owner fails to properly exclude the device, then their network key resides in the device EEPROM after ownership is transferred to a new user. An adversary may use third party Z-Wave markets to acquire collections of network keys. If adversary is able to locate where the originating Z-Wave network is deployed, then they may use the network key to derive the other keys to be able to decrypt, encrypt, and send signed secure frames to the target network.

Replay Attack.

While the embedded application layer command is encrypted in the secure frame, its meaning may be deduced. The source and destination fields of the secure frame

```

PacketNum=3 is an encapsulated encrypted frame
nonceId=0x96 is FOUND in the cache.
Encrypted Payload: 3c 9c f3
Decrypted Payload: 0 62 2
PacketNum=5 is a nonce report frame, Stored at position 1 of the cache.

PacketNum=6 is an encapsulated encrypted frame
nonceId=0x8 is FOUND in the cache.
Encrypted Payload: f 48 2d e5 fa 4d f9 18
Decrypted Payload: 0 62 3 ff 0 0 fe fe
PacketNum=8 is a nonce report frame, Stored at position 2 of the cache.

PacketNum=9 is an encapsulated encrypted frame
nonceId=0xd5 is FOUND in the cache.
Encrypted Payload: 82 67 8a 15
Decrypted Payload: 0 62 1 0
PacketNum=11 is a nonce report frame, Stored at position 3 of the cache.

PacketNum=12 is an encapsulated encrypted frame
nonceId=0x41 is FOUND in the cache.
Encrypted Payload: 3a 7e a7
Decrypted Payload: 0 62 2
PacketNum=14 is a nonce report frame, Stored at position 4 of the cache.

PacketNum=15 is an encapsulated encrypted frame
nonceId=0x3 is FOUND in the cache.
Encrypted Payload: 97 60 ab 16 36 5b a8 ea
Decrypted Payload: 0 62 3 0 0 2 fe fe

```

Figure 53. Decrypted payloads of the secure frames observed as OpenBarley uses the extracted keys to successfully send a secure frame with an unlock door command.

are not encrypted. From [HRRL16], the source and destination devices can be interrogated to learn their supported command classes, manufacturer type, and version information. The conjunction of the device classes, supported command classes, and length of the encrypted payload may be used to make reasonable guesses about a subset of the plaintext. Out of band information, such as visually observing a Z-Wave actuator or invoking a sensor also reduces the entropy of the embedded ciphertext within a secure frame. Consequently, it is reasonable to assume the adversary is able to deduce the meaning of secure frames. For this attack vector, the adversary desires to replay a previously captured frame, such as a door unlock command, to unlock a door.

Z-Wave protects against replay attack with nonces using several policies. First, the recipient generates a new nonce for each secure frame transaction. Second, the

```

0006090: 0000 0000 0000 0000 0000 0000 ff00 0000
00060a0: 0000 0000 0000 0000 0000 0000 0000 0000
00060b0: 5600 1498 eff1 7275 13cc 4200 0000 0000
00060c0: 4232 6402 a801 0000 0000 0000 0000 0000
00060d0: 0000 0000 0000 0000 0000 0000 0000 0000

```

Figure 54. EEPROM of an Aeon Multisensor after being excluded from the network, where the network key is zeroed from memory.

recipient only accepts secure frames if they use a valid nonce as a portion of the transaction IV. Third, a nonce remains valid (i.e., in its cache) until it is used for a received secure frame. Thus, an adversary may attempt to replay an observed secure frame; however, the nonce is no longer valid. While the frame is correctly signed and encrypted, it is ignored by the recipient because the nonce is not in its cache.

Interestingly, none of the three identified replay prevention policies involve the sender nonce. Both OpenBarley and OpenZwave use randomly generated byte strings for their sender nonces, making no attempt to avoid reusing a nonce. Firmware analysis does not reveal the recipient performing validation steps on the sender nonce before copying it into the transaction IV cache. Consequently, no replay protection is provided because the replayed frame contains the correct sender nonce used to signed and encrypt the frame.

An experiment is conducted to verify the act of simply resending an observed secure frame is ignored by the recipient. While the experiment is occurring, the exchanged frames are captured using Wireshark and provided in Figure 55. The captured frames are decrypted using decryptPCAPNG in Figure 56. Spoofing as node ID 100, OpenBarley is used to send a request to the door lock, node ID 30, for its state using a secure frame transaction. The transaction is shown in frames 1-6, where frame 5 is the secure frame. The door lock elicits an observable response with another secure frame transaction in frames 7-12. Following the response, OpenBarley resends frame 5 at frames 13 and 14. The door ignores these frames.

No.	Time	Protocol	Info
1	0.000000000	Zwave	MAC: Singlecast(0) [0xe56194a9 100->30] APP: Security(0x40)
2	0.007053471	Zwave	MAC: ACK(0) [0xe56194a9 30->100]
3	0.262630916	Zwave	MAC: Singlecast(2) [0xe56194a9 30->100] APP: Security(0x80)
4	0.272909165	Zwave	MAC: ACK(2) [0xe56194a9 100->30]
5	1.435424500	Zwave	MAC: Singlecast(2) [0xe56194a9 100->30] APP: Security(0x81)
6	1.440553419	Zwave	MAC: ACK(2) [0xe56194a9 30->100]
7	1.596442550	Zwave	MAC: Singlecast(4) [0xe56194a9 30->100] APP: Security(0x40)
8	1.606608963	Zwave	MAC: ACK(4) [0xe56194a9 100->30]
9	1.617023085	Zwave	MAC: Singlecast(3) [0xe56194a9 100->30] APP: Security(0x80)
10	1.622022096	Zwave	MAC: ACK(3) [0xe56194a9 30->100]
11	1.670594954	Zwave	MAC: Singlecast(5) [0xe56194a9 30->100] APP: Security(0x81)
12	1.680816824	Zwave	MAC: ACK(5) [0xe56194a9 100->30]
13	50.181621763	Zwave	MAC: Singlecast(1) [0xe56194a9 100->30] APP: Security(0x81)
14	53.183773905	Zwave	MAC: Singlecast(1) [0xe56194a9 100->30] APP: Security(0x81)

Figure 55. Frames captured during a replay attack attempt, where requests for the door lock state are ignored.

While the adversary is unable to immediately replay a secure frame once the nonce is used, computer memory is finite. As time progresses, the pool of unused nonces becomes exhausted and eventually a nonce is repeated. An adversary may wait for the target device to reuse the nonce in the previously observed secure frame. At this point, the secure frame may be replayed to the recipient. Since the nonce is in its cache, the frame is accepted. The cost of this attack is a function of the time interval between nonces being reused. Nonces are generated on request and in a sequence dictated by the nonce round key. Evaluating the cost of this attack on Z-Wave devices requires the measurement of the number of nonce requests required before a target nonce is repeated (i.e., a collision event occurs).

Measuring Iterations to Collision.

To efficiently evaluate the expected iterations before nonce collision, the nonce generating algorithm is implemented on a PC with a Linux OS. The initial nonce round key is generated using the OS-provided random device source `/dev/urandom` to derive the target 8-byte nonce. The algorithm continues updating the nonce round key and generating new nonces until the target nonce is encountered. Upon success, the number of iterations required for the collision (i.e., the response variable) is recorded. Because the experiment is randomized, it is repeated several times to

```

PacketNum=3 is a nonce report frame, Stored at po:

PacketNum=5 is an encapsulated encrypted frame
nonceId=0x67 is FOUND in the cache.
Encrypted Payload:  de e3 11
Decrypted Payload:  0 62 2
PacketNum=9 is a nonce report frame, Stored at po:

PacketNum=11 is an encapsulated encrypted frame
nonceId=0x79 is FOUND in the cache.
Encrypted Payload:  d9 24 61 f1 fe 45 4b ff
Decrypted Payload:  0 62 3 0 0 2 fe fe
PacketNum=13 is an encapsulated encrypted frame
nonceId=0x67 is FOUND in the cache.
Encrypted Payload:  de e3 11
Decrypted Payload:  0 62 2
PacketNum=14 is an encapsulated encrypted frame
nonceId=0x67 is FOUND in the cache.
Encrypted Payload:  de e3 11
Decrypted Payload:  0 62 2
PacketNum=15 is an encapsulated encrypted frame
nonceId=0x67 is FOUND in the cache.
Encrypted Payload:  de e3 11
Decrypted Payload:  0 62 2

```

Figure 56. Decryption of the frames captured during a replay attack attempt, where requests for the door lock state are ignored.

derive a sampling of the distribution of the response variable.

Finding at least one 8-byte collision is found incur an excess time penalty, provide suggestive evidence that a brute force nonce collision attack is intractable. To provide supporting evidence, the experiment is adjusted to use nonce lengths where collisions are observable. Four experiments are conducted, while varying the nonce length from one to four bytes. Each experiment is conducted 500 times and split into 50 sample sets of size 10. The sample mean is found for each sample set to derive 50 sample means, which are used to estimate the expected iterations until collision. Figure 57 shows the 99% CI of the true mean number of rounds required for a collision for a given nonce length as error bars in red. The x-axis indicates the nonce length and the y-axis is the response variable in log10 scale.

The expected number of iterations to collision for Z-Wave is plotted against the number of available states for a given nonce size, shown in blue. Ideally, all nonce values should be exhausted before a collision is observed. The figure shows each CI of the measured number of iterations to collision contains the blue line. The response

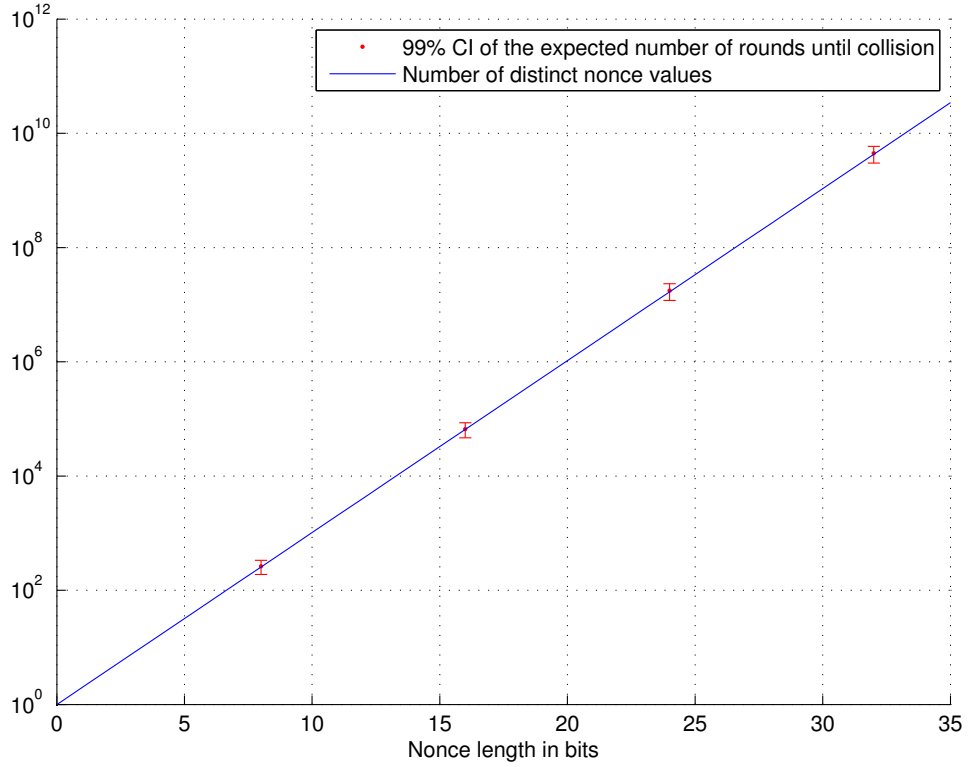


Figure 57. 99% CIs estimating the true mean number of rounds until collision plotted against the number of available states for a given nonce length in bits. Nonce lengths evaluated at 8, 16, 24, and 32-bit.

variable is expected to behave similarly when the nonce length is increased to 64 bits, which is what Z-Wave uses. This is because the nonce length only dictates the number of bits to compare in the 128-bit output block of the cipher. Thus, the expected iterations to collision for the 8 byte nonce case is estimated to be approximately 2^{64} generations. While the *expected* number of iterations falls near the ideal case, the response variable has a unknown random distribution around the mean. The raw data shows cases where it takes more or less than the ideal number of rounds. For cases taking larger number of rounds, nonces other than the target nonce must be repeating. Future work should examine the distribution of the response variable to identify its distribution and determine the probability a given nonce takes a specified

number of rounds before collision.

Nonce Request Rate.

The experiment in Section 4.4 measures the number of rounds required to reach a collision with a randomly derived initial value; however, the response variable is not temporally bounded. While a given number of iterations are feasible to compute offline, as performed in the previous experiment, forcing a device to forward through the expected number of iterations using nonce request messages incurs a time penalty.

An observational study is performed on the Yale door lock and MiCasa Verde Edge controller to determine the maximum rate nonces may be forwarded in nonces per second. To measure the nonce rate for each device, OpenBarley and a SDR are used to repeatedly request nonce values while capturing the frames over a time period. Only one nonce is requested at a time, and requests are repeated if a timeout event occurs. Immediately after receiving a nonce, a new request is made. Table 3 provides the results of the collection.

Table 3. Nonce request rates for a Yale door lock and MiCasa Verde Edge controller.

Device	Num. of Nonces	Duration (Sec.)	Rate (Sec. per Nonce.)
Yale door lock	655	6,066	9
Verde controller	60,354	332,961	6

Table 3 shows the nonce rate to be one nonce every six seconds for the controller and one nonce every nine seconds for the door lock. The reason for the observed difference is because the door lock is battery powered and the controller is supplied through utility. The door lock spends idle moments in a low-power state, which requires a pattern of wake-up signals (i.e., beams) to bring it back to an active state. The controller is always awake, so it may respond faster to requests. The table also shows the collection interval for the controller is approximately four days (i.e., 332,961 seconds) versus less than two hours (i.e., 60,354) for the door lock. The reason is the

door lock batteries became depleted with the repeated interrogation in a few hours. While the batteries could have been replaced to collect more data, an adversary does not have the luxury to replace batteries on a target device.

The data provides evidence indicating the attack method of forcing a nonce collision to successfully replay a secure frame is infeasible. The mean number of rounds before a collision for a 4-byte nonce is approximately 4×10^9 iterations. Given the nonce rates, the average collision time takes approximately 50K years for the controller and 73K years for the door lock. The time for an 8-byte nonce collision is orders of magnitude larger than the 4-byte nonce case. While a brute force replay attack may be infeasible to force a collision, it may certainly be useful as a denial of service attack on battery-powered devices.

4.5 Conclusion

The proprietary Z-Wave security layer provides a mechanism to prevent devices from being manipulated by an external adversary. When configured, a device may accept designated commands received using a secure frame. A secure frame contains a signed and encrypted payload, which is authenticated and decrypted prior to executing the encapsulated command. The work presented herein extends existing reverse engineering efforts to understand and evaluate the proprietary security layer using firmware and network protocol analysis. A comprehensive model of the secure layer is provided, including previously unknown mechanisms including the nonce generating mechanism, an instantiation of the bootstrap key set, and a complete operation key set derivation algorithm. Aspects of the security layer are validated using encryption and authentication tools implementing the known security layer model. To evaluate the effectiveness of the security layer, two attacks vectors are examined. Since the security layer relies on cryptography, several key extraction techniques are realized

and demonstrated on Z-Wave devices. The security layer provides resistance to replay attack, so its aptitude is tested against a brute-force nonce collision attack. The results reveal the attack cost lies in a time frame outside of tractable space.

While conducting the presented research, Sigma Designs made several technical documents available to the public. While the Z-Wave security layer is included in this set [Des16], the specification defines a second, newer version of the security layer. There are two points to make with respect to this specification. First, the new specification does not appear to be compatible with the existing security layer. For example, the document indicates Diffie-Hellman key exchange [Des16] is used to exchange the network key. If the Diffie-Hellman key exchange algorithm is used, then both the controller and device should be observed as exchanging key-generating material with each-other [Res99]. Figure 45 from Section 4.3 clearly shows the controller simply sending the network key to the device (i.e., already generated), which is not consistent with the Diffie-Hellman key exchange algorithm. Second, several of the attack vectors identified herein are briefly discussed, suggesting the publications resulting from the work herein have made an impact on the future design of Z-Wave devices.

Several issues are left as future work. Additional work involves expanding the key extraction attack vector to include extraction of keys in volatile memory. While the keys are stored in XRAM, their location may vary as a function of version, device, vendor, or other factors. Multiple firmware images must be examined to determine if a pattern emerges, where an adversary may be able to guess their location with some additional knowledge about the device. While the nonce generating algorithm is reverse engineered, its model is not verified experimentally. Because a device nonce round key resides in XRAM, a mechanism to extract the key is currently unavailable. While a key leaking attack may work, an easier approach is to modify the firmware with the capability to dump arbitrary memory locations. The capability also permits

additional research into the transaction IV cache policies. Ongoing work regarding volatile memory addresses these issues; the details are provided in Chapter VII.

With regard to replay attack, future work should attempt to characterize the distribution of the random variable for the number of iterations before collision. The distribution permits insight into the probability a brute-force collision attack is achievable in tractable time, such as a few thousand rounds. While the recipient nonce generating algorithm is known, the mechanism used to establish the sender nonce remains unknown. Work should focus on identifying the code region in the firmware to determine if it generates random bytes similar to OpenBarley and OpenZwave, or uses the cipher-based counting mechanism found for recipient nonces. This knowledge may lead to additional vulnerabilities. Finally, when devices are found to use the second security layer, it too should be evaluated for its security aptitude.

V. An Analytical Black Hole Attack Model using a Stochastic Topology Approximation Technique for Reactive Ad-Hoc Routing Protocols

5.1 Introduction

The computer network is a pervasive and critical asset in our society. It permits the sharing of distributed resources to achieve complex social, economic, and scientific objectives irrespective of locality. However, it is also a vulnerability to its hosts because any disruption, degradation, or denial of access to this network adversely affects the objectives of the distributed organization. Moreover, computer networks are subject to disruptive, degrading, and denial attacks at every layer of the network stack [MBC16, ANS13, YM03]. Ad-hoc networks are especially vulnerable to disruption because they rely on coordination *in-situ* rather than apportioning resources *a priori* as done in infrastructure-based networks. Ad-hoc networks provide data routing services to loosely coordinating groups or to address the need for multi-hop communication in environments without infrastructure. With respect to security analysis of systems, there has been significant research on the development of confidentiality and integrity analytical models. Classic models such as the Bell-LaPadula Confidentiality Model [BL73], Lipner’s Integrity Model [Lip82], and the Chinese Wall Model [BN89] have existed for decades. The body of this research has provided a foundational approach to proving security properties in systems under study.

Since networks provide a delivery service, the *availability* security property is of great importance to network designers. Unfortunately, the historical depth of research on analytical availability models is lagging behind that of *confidentiality* and *integrity* research. While there is some availability modeling research in [ZG12, SKB14, AHK08, KSSH09, FW10, Hed09], the *de facto* approach to assessing the effects of availability attacks on networks is measured through simulation. Exam-

ples of this approach include work performed in [SC12, PA12, CJ14, JDB12]. While the simulation approach has a clear utility, there are several drawbacks. First, the approach is exhaustive and scales poorly. High dimensional experiments may take orders of weeks or months to provide conclusions. Second, the simulation models require initial validation when used and revalidation upon any modification to the simulation model-base [AY06]. Third, complex interactions, such as causality, between simulation objects are abstracted and must be statistically estimated through repetition. Effective experiment design can certainly ascertain causality between simulated factors and response variables to generate regression models; however, their mathematical relationship remains hidden.

At the cost of fidelity, these challenges may be avoided by the use of analytical models for security analysis. The contribution of this research is the development of an analytical model for reactive ad-hoc protocols that measures availability degradation of networks subjected to Black Hole attacks. A Black Hole attack is a well-known denial of service attack for ad-hoc networks that deceptively attracts data to flow through nodes under control of an attacker. As packets arrive, they are silently dropped. Such a model can be used in conjunction with other availability models to influence design decisions of distributed system or ad-hoc network developers when limited implementation details exist. Moreover, the model explains the relationship between contributing factors for Black Hole attacks that are hidden when using simulated experimentation. While there has been extensive work in detecting, avoiding, and isolating Black Hole attacks, the relationship between the topology parameters and the attack effectiveness has not been extensively studied. Limited attack effectiveness has been measured using simulation in [PSM11, HQ12, SSD14]; however, the scope of each study is limited to only a few topology types (e.g., protocol, number of Black Holes, number of nodes, and operating area). The results of these observa-

tional studies make it difficult to extrapolate performance for other types of ad-hoc topologies.

In [BM14], a theorem is developed for reactive ad-hoc routing protocols with hop-distance as a primary route metric selection criteria. During route discovery, if a Black Hole node is closer in hop distance than the destination to the source, then the Black Hole node may present a fake route to the downstream nodes with a metric that exceeds the metric of any legitimately proposed route. An analytical Black Hole attack model is developed that calculates the probability of this theorem being true for arbitrary source-destination pairs of a given ad-hoc network. Since the attack probability applies to any route in the network, it is a holistic measure of the susceptibility of a given network to Black Hole attack and may be useful for providing upper or lower bounds of network throughput degradation while under attack.

The analytical model in [BM14] is a function of the number of nodes in the network, the number of nodes conducting Black Hole attacks in the network, and the average node degree of the network. It assumes that the Black Hole nodes are uniform-randomly dispersed within the operating area of the ad-hoc network, and that all nodes within the network have equal probability of being a source or destination of new route. To avoid requiring absolute knowledge of the topology of the ad-hoc network under study, a topology approximation technique is used, seeded by one or more statistically estimated parameters of the topology under study. These parameters include the mean node degree of the network and the number of nodes. They are simpler to estimate prior to the instantiation of an ad-hoc topology than attempting to estimate the graph of the topology instance. The parameters are used to generate a n -ary 2-cube topology, which has average degree n and 2^n nodes [SS88]. The model uses this topology to calculate the probability of attack for all source and destination pairs of the ad-hoc network under study.

The motivation for the utilization of n -ary 2-cubes to approximate ad-hoc topologies is based on the intuition that high-dimensional topologies of an arbitrary orientation, when projected onto a 2-dimensional plane, *appear* as an ad-hoc topology. Moreover, the projection of a single n -ary 2-cube onto a plane may represent a set of flat topologies by rotating the n -ary 2-cube in the higher dimensional space. Unfortunately, the similarities are deceiving, and several issues make it challenging to use this as an approximation method. First, ad-hoc topologies have nodes with varying node degree, whereas the node degree for all nodes in a n -ary 2-cube have a constant degree. This means that as the variance of the node degree of an ad-hoc network increases, the approximation will not be able to represent portions of the network with extreme connectivity. Second, given an ad-hoc topology with a known node degree and number of nodes, it is likely that, due to the ridged definition of the n -ary 2-cube, a corresponding n -ary 2-cube approximation having the same values for both parameters does not exist. Any application using this approximation, including the Black Hole attack model, must perform a trade-off study to determine which n -ary 2-cube approximation minimizes calculation error, degrading the utility of the approximation. Third, ad-hoc topologies may become partitioned over their lifetime due to mobility or node failure. A n -ary 2-cube is unable to model network partitions because the node degree of the approximation is homogeneous within the topology. Fourth, two nodes may be within transmission distance within the projection of an arbitrarily rotated n -ary 2-cube onto a 2-dimensional plane; however, their Euclidean distance in the high dimensional space may be beyond transmission range. This means that the projection will contain edges that do not exist in the n -ary 2-cube.

The work presented in this chapter extends the work accomplished in [BM14] while addressing the disadvantages of using an n -ary 2-cube as an approximation technique. First, a simple simulated experiment is conducted to enhance the credibility of the

theorems derived in [BM14]. Second, the Black Hole attack model is generalized for arbitrary network topologies. In this generalization, the topology state is known and a simulated experiment is conducted to show that the analytical model is able to predict the network level effects of Black Hole attack. Third, the generalized model is extended to incorporate unique aspects of ad-hoc networks; namely, that nodes may become partitioned and that the true topology state is difficult to realize prior to its instantiation. The n-ary 2-cube topology approximation is replaced by a set of prototype *neighborhoods*, derived statistically via k-means clustering. A third simulated experiment is conducted to validate the extended analytical Black Hole attack model. To illustrate the improvement of the analytical model derived in this work, it is compared with performance predictions using the original model defined in [BM14].

Ad-Hoc Network Routing Background.

An ad-hoc routing service is comprised of four core components: 1) determining topology state, 2) calculating routes, 3) selecting a route, and 4) forwarding packets according to the selected route [SYJ10]. The predominant challenge for the routing service is to efficiently realize and maintain the state of network topology while contending with confounding dynamics in the physical, RF, and logical domains. Examples of dynamic events include node power failures, RF interference, and topology discovery on initial deployment. Global awareness of these events is achieved through protocol coordination, through which participants discover and exchange local topology state information with peers to identify potential routes. The fittest route is selected from this state update per destination or as needed.

One major aspect of routing protocols is when routes are calculated. A *proactive* protocol will enforce a periodic synchronization between all nodes to achieve topology

state coherency. A node with fresh topology information can immediately calculate the next hop in the forwarding path or, depending on the protocol, determine the complete route. To minimize coordination overhead, *reactive* routing protocols only coordinate when necessary. The trade-off between reactive and proactive strategies is route setup time and effective bandwidth. Proactive protocols have more deterministic route setup times at a cost of utilizing higher bandwidth [QZJX09]. Reactive protocols utilize less bandwidth for control packets, but have higher variance in the route setup period [KL07].

This research generally applies to reactive ad-hoc protocols; however, the specific work focuses exclusively on Ad-Hoc On-demand Distance Vector routing (AODV) and Dynamic Source Routing (DSR). Both of these have matured to be de facto reactive protocols and constitute a base design class from which other reactive protocols have been derived. In essence, to attempt a unified analysis, this work applies to the common aspects of all reactive protocols (i.e., the route discovery process) and, more specifically, to AODV and DSR.

Ad-Hoc On-demand Distance Vector Routing.

AODV routing is a reactive routing, forward updating, hop-by-hop, flat, and single-path routing protocol [PBRD03]. The protocol is broken into three services: 1) Route Discovery, 2) Route Repair, and 3) Packet Forwarding. The Route Discovery process occurs when a source node desires to route to a destination. The source node sends a route request (RREQ) packet that is flooded throughout the network via broadcast. As the RREQ is propagating the network, intermediate nodes append their ID to the RREQ and store a forwarding rule towards the source in preparation of a reply message. When the destination (or an intermediate node that knows the forwarding path to the destination) receives a RREQ it responds with a route reply

(RREP) message. The responding node places the path information collected during the RREQ into the RREP and sends it along the reverse path previously established during the RREQ flood to the source node. Each node that receives the RREP adds the forwarding rule to their route table and forwards the RREP toward the source. Because of the flooding nature of a RREQ, the destination generates a RREP for each discovered path.

Intermediate and source nodes, receiving a RREQ, RREP, or a route error (RERR) message, update their forwarding table if 1) the destination sequence number in the coordination message is higher than the one stored in their table, or 2) the destination sequence number is the same as the entry in its routing table, but the hop-count in the message is shorter. Coordination messages containing higher destination sequence numbers imply fresher routing information.

Packet forwarding is achieved via a distance vector table stored at each node in the network containing entries for each known destination. For each destination, the node stores the next hop, distance in hops to the destination, and the sequence number of the latest update to the route. When application packets arrive to be forwarded, the node examines the destination in the packet and determines the next hop using the appropriate entry in the routing table.

Dynamic Source Routing.

DSR is a reactive, forward updating, source-based, and flat routing protocol [JHM07]. Its route discovery and maintenance behavior is very similar to AODV; however, the major differences between the protocols are the manner of packet routing and how the routes are stored. Unlike AODV, the route is maintained completely by the source node in DSR. The source node is responsible for generating the route request and has complete freedom to select any route reply to use when routing pack-

ets. Instead of storing the route hop-by-hop, DSR uses source routing where the source node places complete routing information in each application packet. Each intermediate node along a route uses this information to determine the next hop. The intent of the designers is to follow the analog of the TCP/IP *fate sharing* [Cla88] by placing the majority of the complexity burden at the end nodes.

The route selection method is not specified in [JHM07], but rather, is left up to the implementation of the protocol. Many DSR implementations use hop count as the route selection metric, such as Network Simulator 2, PicoNet, and the Rice Monarch Project.

Related Work.

A significant number of published research papers measure simulated Black Hole attacks on ad-hoc networks. Performance degradation from Black Hole attack is measured on AODV networks in [SC12, PA12, CJ14, PSM11, BBS09]. For DSR, performance results can be found in [SSD12, AS11, JPY⁺09, BS09] . Performance measurements of Black Hole attacks in other networks types include [JDB12, HQ12, GBHF01]. Due to the large parameter space of ad-hoc networks, there is little overlap between each study; however, they all indicate that Black Hole attack decreases network throughput.

Several recent works develop analytical models to characterize the performance of Black Hole attacks. In [ZG12], a probabilistic model is developed to quantify the effects of a Black Hole attack in a smart grid network. In [CTK16], an Adaptive Neural-Fuzzy Inference System (ANFIS) is created to detect Black Hole node behaviors. A Colored Petri-Net model is developed in [HQ12]; however, the model must be simulated to derive results. The foundations of a formal Black Hole attack model for wireless sensor network routing protocols is proposed in [SKB14].

In [AHK08] an analytical model is presented to calculate network throughput under denial of service attacks; specifically, these attacks are Jellyfish and Black Hole. Their model estimates the availability of a network flow (i.e., a group of packets traversing a route) based the proportion of lifetime that a network flow incurs zero throughput. The expected time a route has zero throughput is calculated as the product of the probability that at least one malicious node is in an arbitrary route of a certain length and the expected correction time to expunge all malicious nodes from that route. The correction time is based on the number of attempts to detect the attack, rediscover an alternative route, and repair for each malicious node in the route. Given the expected interval of zero throughput, one can calculate availability as one minus the ratio of the time of zero throughput over the expected duration of the flow.

A completely different Black Hole model is proposed in [KSSH09] to model packet loss instead of throughput of an AODV Mobile Ad-hoc Network (MANET) during a Black Hole attack. König assumes uniform node density to simplify the topology so that the number of nodes included in a given topology search area may be determined. By letting the radius of this search area be a multiple of the transmission distance, one can geometrically determine the number of nodes reachable at each hop. The author of this method also makes an attempt to address the border effect errors with the uniform density assumption; however, the authors acknowledge that their method is imperfect. Given the set of nodes included in the i^{th} RREQ of the expanding ring search, König can find 1) the probability that at least one Black Hole is in the search area and 2) the probability that the destination is within the search area. By taking the product of these two probabilities for each phase of the ring search, the sum of products is the probability an arbitrary route in a network with a given network density is subject to Black Hole attack.

5.2 Revisiting the Hypercube Black Hole Attack Model

From [BM14], Black Hole node b in network topology G is able to provide a false route with a winning metric to drop application packets on a route from source node s to destination node d if:

$$\exists b \in B \quad s.t. \{h(s, d) > h(s, b)\}, \quad (1)$$

where $h(x, y)$ is the minimum hop distance between x and y in network topology G and B is the set of Black Hole nodes present in G . While a proof is provided in [BM14], the prior work did not provide experimental validation.

Validation of the Attack Condition.

A series of simulated experiments are conducted to test the validity of Equation (1) by observing the effects of packet loss while varying the hop distances between the source, destination, and a Black Hole node over a linear topology. The use of a linear topology allows explicit control of the hop distances of each player in the experiment series. The linear topology consists of 21 nodes, where a single source node is placed in the center of the network and is flanked by 10 nodes on each side. For a given simulation experiment, the destination node is designated as one of the 10 nodes on the right of the source node and a Black Hole node is designated as one of the 10 nodes on the left. During the simulation, the source node attempts to establish a route to the destination. The Black Hole node participates during the route setup and attempts to move the route through itself. Once the route is selected, the source sends constant bit rate (CBR) traffic to the destination. If the Black Hole attack is successful, none of the packets reach the destination because they are being forwarded to the opposite side of the network to the Black Hole node, which drops all of them.

Packet loss is recorded from 100 scenarios generated by testing all combinations of $h(s, b)$ and $h(s, d)$, where each hop distance takes on a value from 1 to 10. Each scenario is replicated 100 times to generate a mean normalized packet loss statistic, where normalized packet loss is the proportion of packets lost over the total number of sent packets. The entire sequence of experiments are conducted using both AODV and DSR protocols.

Each wireless ad-hoc node in the network is a simulation model, which is comprised of an antenna, radio, propagation model, and a protocol stack in Network Simulator 2.34 (ns-2.34). Specifically, the stack is comprised of an omni-directional antenna with unity gain, a 914MHz Lucent WaveLAN Direct Sequence Spread Spectrum (DSSS) radio, an implementation of IEEE 802.11 MAC layer, and a reactive MANET routing protocol. The stack enables each node to provide packet routing for the wireless ad-hoc network. Besides basic routing services, some nodes are designated as application end-points, which send or receive CBR traffic. The linear topology is enforced by placing nodes in a line, where the transmission coverage area of each node contains a either two neighbors, or for the end nodes, a single neighbor. A Black Hole is a node with a modified MANET routing protocol designed to conduct Black Hole attacks and is identical to the simulation model used in [BM14]. The simulation transmission range of each radio is 250 meters.

The observed normalized packet loss for AODV and DSR as a function of destination and Black Hole hop distances are shown as a surface plot in Figure 58. A given point in the z axis is the normalized average packet loss observed when $h(s, b) = x$ and $h(s, d) = y$. The figure shows that in all cases where $h(s, b) \geq h(s, d)$, the packet loss is zero. Conversely, packet loss is incurred for each case where $h(s, b) < h(s, d)$. Packet loss is at 100% when $h(s, d) - h(s, b) > 1$. The reason that packet loss is only at 80% when $h(s, d) - h(s, b) = 1$ (i.e., the Black Hole node is one hop closer

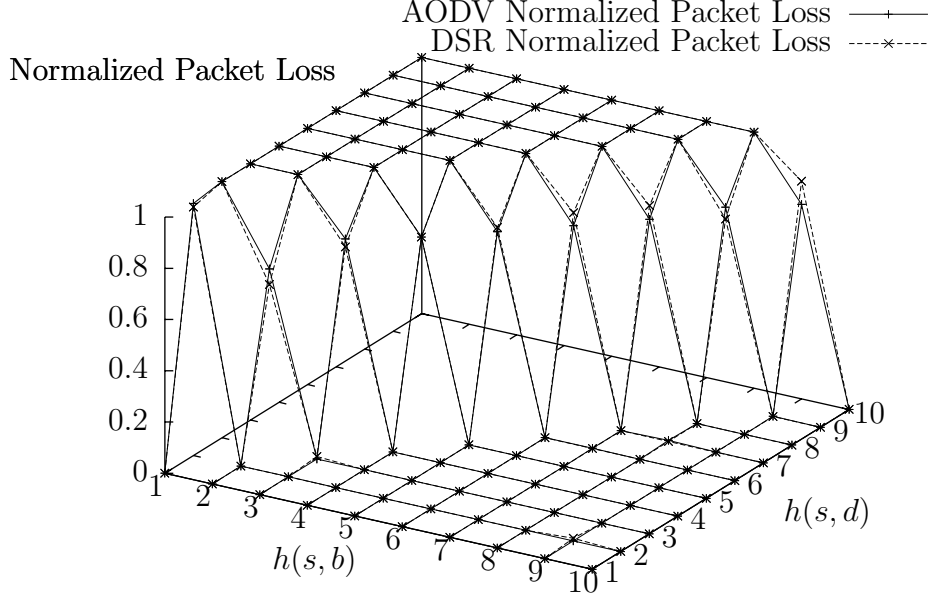


Figure 58. Normalized packet loss as a function of hop-distance

than the destination node to the source) is an artifact of the simulation models for AODV and DSR in ns-2.34. Recall that from [BM14], the shortest path a Black Hole can advertise to the source node without masquerading as the destination has length $h(s, b) + 1$. When $h(s, d) - h(s, b) = 1$ both the Black Hole and destination node will respond with routes having the same hop count metric. As a tiebreaker, the RREP arriving first is selected. Since $h(s, d) - h(s, b) = 1$, the RREP from the Black Hole node has fewer hops to traverse than the RREP created from the destination node. However, the simulation avoids RREQ broadcast collisions at each hop by waiting a uniformly random time period before rebroadcasting. When the destination and Black Hole hop distances to the source node are close, there are occurrences where the Black Hole node receives the RREQ at a later time than the destination because the predecessors of the Black Hole node encounter a larger cumulative random delay.

Decoupling the Hypercube Topology from the Model.

The Analytical Black Hole model is the probability Equation (1) is true for all source destination pairs. This is calculated by considering all possible relative distances of $h(s, b)$ and $h(s, d)$ in a given network. From [BM14], the discrete probability of this event is

$$P(A) = \sum_{h=1}^n P(A|H = h)P(H = h), \quad (2)$$

where $P(A)$ is the probability of a Black Hole attack and $P(H)$ is the probability that $h(s, d) = h$. Given $h(s, d)$, the probability of a Black Hole attack is simply the probability that at least one of the B Black Hole nodes are closer than h hops to the source. This requires knowing the number of possible neighbors that are closer than h and finding the probability that at least one of the neighbors is a Black Hole node. The n -ary 2-cube topology is useful here because this quantity can be derived analytically given parameter n , which is the longest expected route length. Moreover, the symmetric properties of a n -ary 2-cube topology result in every node having the same quantity of unlabeled neighbors at each hop distance. This allows $P(A)$ to be calculated without considering the relative location of each source node within the topology. Let $q(x)$ be the quantity of neighbors including Black Hole nodes at hop distance x . The number of nodes that are closer than h hops is the sum of $q(x)$ for all $x = 1, 2, \dots, h - 1$. Given $q(x)$ is known for all values of x , N is the number of nodes in the topology, B is the number of Black Hole nodes in the network, then $P(A)$ is a hyper-geometric discrete random variable shown in Equation (3).

$$P(A) = \sum_{h=1}^n \left\{ \left(1 - \frac{\binom{(N-2) - \sum_{i=1}^{h-1} q(i)}{B}}{\binom{N-2}{B}} \right) \frac{q(h)}{N-1} \right\}. \quad (3)$$

Because this analytical model is derived for n-ary 2-cubes, the model assumes that all source nodes have the same $q(x)$ function. Decoupling the analytical model from the n-ary 2-cube topology requires that $q(x)$ is context dependent on the position of the source node within the network being analyzed. If the topology is known, then the number of $q(x)$ functions has an upper bound of N , implying that each node has a unique $q(x)$ function. Moreover, the conjecture is that the set of these $q(x)$ functions is sufficient to describe the network topology under study. If a topology exhibits symmetry, then there will consequently be duplicate $q(x)$ functions describing the network. Ignoring duplicate functions, fewer $q(x)$ functions are required to describe the symmetric topology. Let a *source node class* be a set of one or more nodes that share the same $q(x)$ function. More specifically, source node class C_k has $q_i(x) = q_j(x) \forall i, j \in C_k; x = 1, 2, \dots, n$. When a route discovery is initiated, there is an associative probability that the source node originating the discovery belongs to a particular source node class. The source node class membership probability equation is simply the proportion of nodes in a class over the number of nodes in the network, where $|C_k|$ is the cardinality of class C_k . This is

$$P(s \in C_k) = \frac{|C_k|}{N} \forall C_k, k = 1, 2, \dots, K. \quad (4)$$

Incorporating Equation (4) into the model results in

$$P(A) = \sum_{k=1}^K \sum_{h=1}^n P(A|H = h) P(H = h|s \in C_k) P(s \in C_k) \quad (5)$$

Note that K is the number of source node classes for the network and its value is a function of the topology under study. Expanding Equation (5) is

$$P(A) = \sum_{k=1}^K \sum_{h=1}^n \left\{ \left(1 - \frac{\binom{(N-2) - \sum_{i=1}^{h-1} q_k(i)}{B}}{\binom{N-2}{B}} \right) \frac{q_k(h)|C_k|}{N(N-1)} \right\} \quad (6)$$

Validation of the Generalized Attack Model.

Equation (6) is validated through a series of simulated experiments. To illustrate the concept of source node classes and $q(x)$ functions, these experiments use a simple fixed 4x4 grid network. To keep the topology fixed at 16 nodes and in a 4x4 grid structure, non-Black Hole nodes are removed from the network as Black Hole nodes are added. This keeps the number of source node classes and $q(x)$ functions constant. The distances between nodes force communication between only cardinally adjacent nodes in the grid. A 4x4 grid topology has three source node classes, labeled C_1 : *Corner Nodes*, C_2 : *Outer Edge Nodes*, and C_3 : *Inner Nodes*. Figure 59 shows the topology with each node assigned to one of the three source node classes. From the figure there are four corner nodes in C_1 , eight outer nodes in C_2 , and four inner nodes in C_3 respectively. The $q_k(x)$ function (i.e., the quantity of neighbors x hops from a source node in class k) for each source node class is shown in Table 4.

Table 4. Values for $q_k(x)$ for each source node class in the 4x4 grid

Hop Distance	$x =$	1	2	3	4	5	6
Corner Nodes	$q_1(x) =$	2	3	4	3	2	1
Outer Edge Nodes	$q_2(x) =$	3	4	4	3	1	0
Inner Nodes	$q_3(x) =$	4	6	4	1	0	0

The normalized packet loss is observed as the number of Black Hole nodes are increased in the topology from one to eight. For each scenario, 50 4x4 grid topologies are generated. For each topology a subset of the 16 nodes are randomly designated as Black Hole nodes and 100 source-destination pairs are also randomly designated. Each connection pair is independently simulated, where the source attempts to establish

a route with the destination in the presence of Black Hole nodes. Once the route exists, the source sends CBR traffic to the destination. Each simulation is repeated 10 times to account for the random packet delay incurred during the simulation and to avoid confounding effects of congestion and route caching, which are not currently accounted for in the analytical model. Statistics on the number of dropped, sent, and received packets are collected and used to estimate the mean normalized packet loss for each factor level combination.

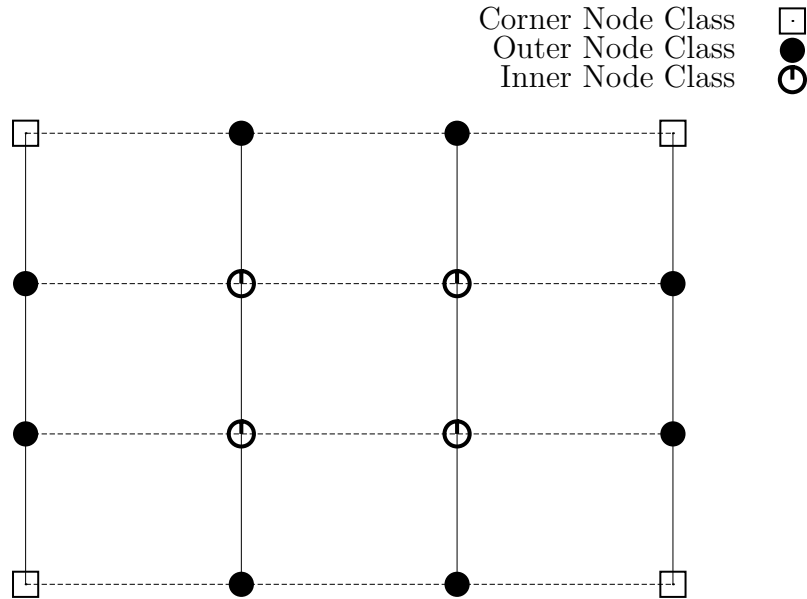


Figure 59. 4x4 Wireless grid topology with labeled source node classes

The probability of attack is calculated using Equation (6) and is overlaid with the simulation results in Figure 60. Clearly the analytical model's predications are within the 95% CIs for all Black Hole levels for both protocols. The figure also shows that the attacker experiences diminishing returns as the number of Black Hole nodes grows, with an upper-bound at approximately 78% normalized packet loss. An attacker may use this curve to optimize cost of placement versus payoff. Using this network as an example, approximately three quarters of the maximum performance is achieved by

deploying at least three Black Hole nodes. In terms of security defense analysis, the expected packet loss does not exceed 80%. System designers may use this upper-limit to implement distributed applications that tolerate operating conditions, such as through caching, redundancy, or multipath.

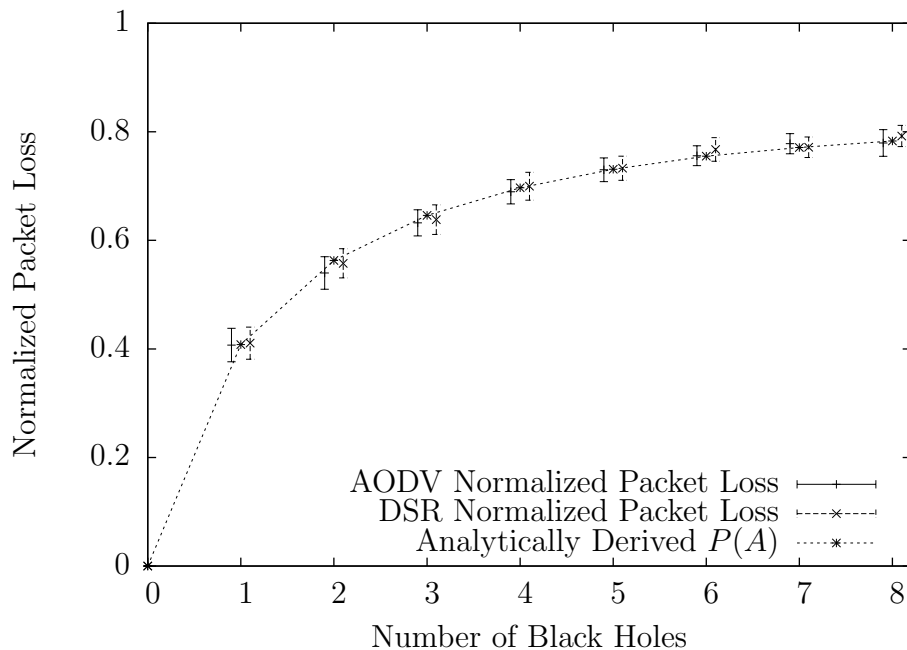


Figure 60. Normalized packet loss due to Black Hole attack for the 4x4 grid network

5.3 The Attack Model Adapted for Ad-hoc Networks

The analytical model presented in the previous section is generalized to account for any topology that may be described as a set of source node classes, each with a particular $q(x)$ function. However, there are several challenges to overcome so that this analytical model may be useful for ad-hoc networks while avoiding the n-ary 2-cube approximation technique. First, the topology is not known *apriori*, making it difficult to assess the susceptibility of a network to Black Hole attack until the topology is instantiated. Second, ad-hoc networks may be partitioned, which violates the assumptions of the original model. In this section, the analytical model

is enhanced to address these two challenges.

Describing Ad-hoc Topologies Stochastically.

In [BE03], the authors stochastically generate random topologies using a small set of parameters to empirically derive neighbor hop distance probability density functions for a variety of ad-hoc network types. This work expands on [BE03] by using K-means clustering on stochastically generated topologies to identify K distinct source node classes and the associative $q_k(x)$ functions (i.e., neighbor hop distance probability density functions) to represent an expected configuration of the ad-hoc network prior to its existence. Using this approach, only the deployment strategy, number of nodes, and operating area are required *a priori*. Unlike the case for known topologies, the number of classes is not bounded by the number of nodes N . Because the network may take on a variety of topology configurations, K may be orders of magnitude larger than N , which implies that a larger set of $q(x)$ functions are required to adequately describe an ad-hoc network than a particular topology instantiation such as the 4x4 grid topology, described in a previous section.

The strategy of this approach is to generate a set of source node classes that adequately describe the distribution of possible topology instantiations. This is accomplished stochastically by generating M random topology instances according to the expected number of nodes, area of deployment, and deployment strategy. For each of the M topology instances, the $q(x)$ function of every node is empirically derived, resulting in a collection of $M \times N$ distinct $q(x)$ samples. The $q(x)$ function samples are associated using K -means clustering [SS07]. This results in the identification of K source node classes, where the center of each class C_k is an n -dimensional vector of hop-distance quantities (i.e., $q_k(x)$). For stochastic topologies, the probability that source node s belongs to a particular source node class C_k is estimated as the pro-

portion of the generated $q(x)$ samples in cluster k over the total number of generated $q(x)$ samples. The update to Equation (4) is shown in Equation (7).

$$P(s \in C_k) = \frac{|C_k|}{MN}, \quad (7)$$

where N is the number of nodes and M is the number of instantiated topologies.

Network Partitioning.

Given the area, number of nodes, and deployment strategy, there is some probability that ρ nodes will be partitioned due to spatial separation. To account for partitioned nodes, let ρ_k be the average number of partitioned nodes for source node class C_k . This statistic can be derived using $q_k(x)$, where

$$\rho_k = N - \sum_{h=1}^n q_k(h).$$

When a destination node is partitioned from a source node, the existence of any Black Hole in the same network partition as the source node results in a Black Hole attack. Moreover, a route is established and additional network traffic is generated that would otherwise have not existed. The effect is that on a pair-wise comparison, partitioned networks with Black Hole nodes will not only have increased packet loss; they will also have an increase in the number of sent packets. Let ψ be a random event where the source attempts to connect with a destination that is one of the ρ nodes partitioned from the source node. With respect to a given source node class C_k , the probability a randomly selected destination is partitioned is

$$P(\psi_k) = \frac{\rho_k}{N - 1}.$$

Given that the destination node is partitioned from the source node $s \in C_k$, the

source node will only receive replies from Black Hole nodes during route discovery. Therefore, if there is at least one Black Hole node that is not partitioned from the source, then a Black Hole attack occurs. The equation for this is

$$P(A|\psi_k) = \begin{cases} B < \rho_k, & \left(1 - \frac{\binom{\rho_k-1}{B}}{\binom{N-2}{B}}\right) \\ B \geq \rho_k, & 1 \end{cases} \quad (8)$$

Note that when there are greater or equal number of Black Holes nodes than the expected number of partition nodes, then at least one Black Hole node must exist in the same partition as the source node.

Stochastic Analytical Black Hole Attack Model for Ad-Hoc Networks.

Considering the stochastic representation of the ad-hoc topology and accounting for partitioning, the revised form of the analytical model for Black Hole attack on ad-hoc networks is

$$P(A) = \sum_{k=1}^K \left\{ \left[P(A|\psi_k)P(\psi_k) + P(A|\psi'_k)(1 - P(\psi_k)) \right] P(s \in C_k) \right\} \quad (9)$$

The probability of attack is the sum of the attack probabilities for all source node classes. For each class C_k , the attack probability accounts for events including partitioned destinations and non-partitions with probabilities $P(\psi_k)$ and $(1 - P(\psi_k))$ respectively. Since $P(A|\psi_k)$ is defined in Equation (8), this leaves $P(A|\psi'_k)$ to be defined, which is

$$\begin{aligned} P(A|\psi'_k) &= \sum_{h=1}^n P(A|H)P(H = h) \\ &= \sum_{h=1}^n \left[\left(1 - \frac{\binom{N-2-\sum_{i=1}^{h-1} q_k(i)}{B}}{\binom{N-2}{B}} \right) \frac{q_k(h)}{N-\rho_k-1} \right]. \end{aligned} \quad (10)$$

Table 5. Four simulated ad-hoc network types under test

Type	Nodes	Area (m^2)	Density (m^{-2})
1	10	500	0.02
2	20	500	0.04
3	20	1000	0.02
4	40	1000	0.04

Equation (10) is derived from Equation (3) and adjusted to account for partitioned nodes. With Equations (8), (10) and for completeness, the analytical model expression is expanded to

$$P(A) = \sum_{k=1}^K \left\{ \left[P(A|\psi_k) \frac{\rho_k}{N-1} + \left(1 - \frac{\rho_k}{N-1} \right) P(A|\psi'_k) \right] \frac{|C_k|}{MN} \right\} \quad (11)$$

Validation of Revised Model.

An experiment is conducted via simulation to validate the revised analytical Black Hole attack model for ad-hoc networks. Four distinct topology types are chosen to represent a sample space of ad-hoc topologies that vary in operating area, number of nodes, and the resulting density. The properties of each topology type are described in Table 5.

Each topology type has its own set of source node classes, which are found using the population sampling and K -means clustering method described earlier in this chapter. One thousand random topologies are generated to create large sample sizes for each topology type. K -means clustering is applied to each sample set, where $K = 200$ is found by analyzing the change in variance as K increases [Tho53]. The experiment is a full factorial design with factors of topology type, number of Black Hole nodes (1 to 10), and ad-hoc network protocol (AODV and DSR), resulting in 80 distinct factor-level combinations. For each factor-level combination, 100 topology instances are generated and simulated independently. For each topology instance, 100 randomly selected source-destination pairs attempt to establish routes and transmit

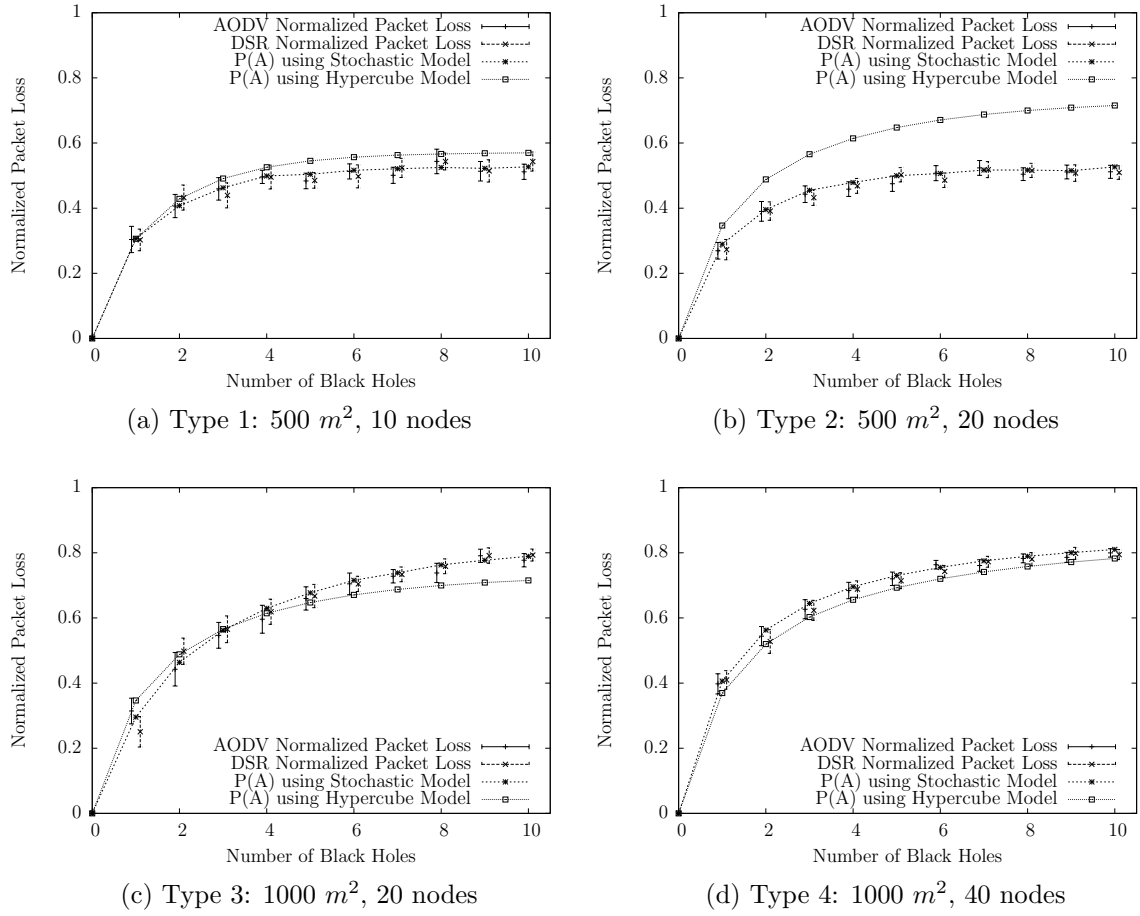


Figure 61. Simulated packet loss of ad-hoc networks vs. hypercube and stochastic analytical model predictions

CBR traffic in the presence of Black Hole nodes. The quantity of replications is selected to minimize sampling bias in the results. The packet loss for each connection is recorded and used to derive an estimate of the expected packet loss for the factor-level combination. Because the analytical model does not account for congestion or mobility dynamics, they are not simulated in this experiment to avoid measuring confounding factors.

Results and Analysis.

The results of the experiment are shown in Figure 61. For each topology type, the measured 95% CI of the normalized packet loss for AODV and DSR is plotted against analytical results calculated using Equation (11). The analytical model derived in [BM14] is also plotted to evaluate the benefits of the enhancements to this model. For this case, the hypercube with the closest number of neighbors is used as the topology approximation model. With respect to the figure, the x axis indicates the number of Black Hole nodes deployed into the network for a given normalized packet loss response.

The results show that the stochastic analytical model's prediction of packet loss falls within the 95% CI for all scenarios for each topology type. This is strong evidence in support of the claim that $P(A)$, as calculated by the revised analytical model, can be used to predict normalized packet loss of a network under Black Hole attack. Moreover, the original hypercube analytical model performs poorer than the stochastic model for the ad-hoc topology scenarios under study. In Figures 61a and 61d, the hypercube topology estimates are tolerable, but in several cases the analytically derived performance values are under or over estimating the simulation results. The hypercube topology approximation does not predict normalized packet loss for ad-hoc topology types 2 and 3. In Figure 61b the hypercube model significantly overestimates Black Hole attack. The hypercube performance curve in Figure 61c both under-estimates and over-estimates performance. Excluding the cases where there are zero Black Holes, the hypercube model prediction falls within the 95% CI packet loss in only nine of the 40 remaining data points presented in Figure 61.

Another noticeable difference between the performance predictions of the two models is that the stochastic curve has some slight variation between data-points while the hypercube performance curves do not. This is because the source node

classes are derived statistically and consequently incur a degree of sample variation in the K class $q(x)$ functions. On the other hand, the hypercube analytical model uses the n -ary 2-cube topology, so it has no amount of variation in its single source node class $q(x)$ function.

5.4 Conclusion

This work provides a network availability model to be used to assess the impact of network disruption due to Black Hole attacks for insecure reactive ad-hoc protocols in ad-hoc topologies. Given the downsides to using a hypercube topology, the model is revised and a series of experiments are performed to validate these revisions.

First, the attack condition is validated using an experiment where a linear topology is used to enforce a variety of hop distances between a source, destination, and Black Hole node. The results confirm the effectiveness of a Black Hole attack is based on the relative hop distances.

Second, rather than utilize the hypercube approximation technique, the analytical model is generalized to use arbitrary topologies for calculating the attack probability, where a topology is represented as a set of source node classes, each with a unique $q(x)$ function. A given $q(x)$ function represents an existential pattern within the topology that describes the number of neighbors at a given hop distance x . A second experiment is conducted using simulation for a grid network to validate the generalization of the analytical model. A 4x4 grid topology is used because the topology is deterministic and requires only three source node classes to represent the entire topology of 16 nodes. The experimental results show that the analytical model is able to utilize the $q(x)$ functions of the three source node classes to predict the mean normalized packet loss of the topology as a function of the number of Black Hole nodes in the topology.

Third, the analytical model is extended to account for specific aspects of ad-hoc topologies. Because the topology is not known until it is instantiated, the source node classes cannot be explicitly realized. Instead, they are statistically derived using K-means clustering on a large sample of instantiated topologies having the same number of nodes, operating area, and deployment strategy. The ad-hoc topology may contain zero or more network partitions. In this work, the model is extended to account for cases where the source node is partitioned from a destination node. A third experiment is conducted through simulation to validate the ad-hoc network adaptations to the model. Four different types of networks are studied by varying the number of nodes and operating area. The results show that the additions to the analytical model aid it in predicting the impact of Black Hole attacks on ad-hoc networks. Moreover, the experiment shows that the revised model provides better prediction of mean normalized packet loss than using the original hypercube model. For this experiment, the hypercube model correctly predicts 9 out of 40 scenarios, whereas the stochastic analytical model correctly predicts measured normalized packet loss for all 40 scenarios, suggesting a significant improvement in the model.

Given these accomplishments, there are several areas identified as future work. First, to minimize confounding effects and measure fundamental Black Hole attack response, the significant aspects of congestion and mobility have been avoided. With the foundational results of this research, future work should examine these aspects; the analytical work in [HXSGLA10] may provide a starting point. Second, there are several varieties of Black Hole attacks and many other types of network disruption attacks. The theory and model presented in this chapter address a single type of Black Hole attack. This work can be readily extended to address other variants of the attack such as commandeering, masquerading, and wormholes.

VI. The Z-Wave Routing Protocol and its Security Implications

6.1 Introduction

Z-Wave is an implementation of a complete IoT substrate, containing well-defined communication, networking, and application layer protocols. The user combines Z-Wave capable sensors, actuators, controllers, routers, and Internet gateways to provide home and office automation services. Example automation services include security monitoring, smart power management, and climate control; still, the device-level granularity of system composition allows the user to realize more exotic IoT services. The communication substrate uses a proprietary radio stack, where the PHY layer and MAC layer are defined in [ITU12a]. Additionally, a significant portion of the application layer is revealed in OpenZwave source code [Ope15b]. Sitting between the MAC and application layer is the optional network layer, which handles multihop routing.

While the specifications of the PHY, MAC, and a portion of the application layer of the Z-Wave protocol stack are publicly accessible, few details regarding the network layer exist within the public domain. The most significant reference is found in [Pae13], where the author indicates that Z-Wave uses *static source routing*. Moreover, routes are calculated from a centralized routing table and embedded into routed messages to dictate their forwarding behavior [Pae13]. In [FRRP17], the basic Source Route (SR) forwarding mechanism for Z-Wave is described. Beyond these sources, several non-peer-reviewed blogs are available that provide supporting details on the protocol [Zwa16, rw15]. While they correlate with [Pae13], they do not provide new information about the routing protocol. To date, an open source implementation of Z-Wave routing does not exist. While OpenZwave is an open source implementation

of a Z-Wave controller for a PC, the routing logic resides in the firmware of the required USB Z-Wave transceiver dongle.

The contributions of the work herein are as follows. First, a significant portion of the Z-Wave network routing protocol is reverse engineered to include a detailed understanding of the frame forwarding and topology management mechanisms. Second, the aspects of these mechanisms are analyzed to identify source and data integrity vulnerabilities of the protocol. The vulnerabilities are exploited to conduct a Black Hole attack on a real-world Z-Wave network. Based on the vulnerabilities discovered on the network under study, several recommendations are provided to enhance the security of the routing protocol.

The remainder of this chapter is as follows. Section 6.2 provides a brief description of the salient aspects of Z-Wave and related work that is relevant to the chapter. Section 6.3 describes the data acquisition, exploitation, and analysis methodology used for reverse engineering and performing a security assessment of the protocol. The reverse engineering results are composed in Section 6.4 to provide the most complete and publicly available model of the Z-Wave routing protocol. Section 6.5 provides the results of the security assessment, including existing security mechanisms offering some resistance to source and data integrity attacks. In the presence of the existing security mechanisms, a Black Hole attack is conducted to demonstrate the synthesis of discovered vulnerabilities on a Z-Wave network. Several recommendations to the protocol are included in this section. This is followed by Section 6.6, which provides considerations for future work and conclusions.

6.2 Z-Wave Background

Using the Z-Wave physical layer, messages are asynchronously exchanged over the RF medium as MPDU frames. A MPDU contains a header, consisting of iden-

tification and control fields [ITU12a]. The MPDU payload contains data pertaining to an application layer command, query, or report. Optional layers exist between the MPDU header and application layer, including the network and security layers. Several other aspects of Z-Wave are provided below.

Pairing Operation.

A distinguishing aspect of Z-Wave is the manner in which devices enter and leave the network [Pae13]. This *inclusion* or *pairing* process is similar to Bluetooth device pairing. A user wishing to add a device to the network first puts the Z-Wave controller and the new device into a pairing mode. While placing a device in pairing mode is device specific, this is commonly achieved by pressing a button or physically resetting the device. While in the pairing mode, the controller adds any device found to also be in pairing mode. Depending on the controller implementation, it may present a list of discovered devices and allow the user to select ones to add. Alternatively, the controller may add all discovered devices within physical proximity without discrimination. Examples of each type of controller include the Mi Casa Verde Vera [Ver17] and Aeon Labs Z-Stick [Z-S16], respectively. Removing a device from a Z-Wave network is accomplished in a similar manner.

Node Identification.

From [Pae13, ITU12a], Z-Wave devices are identified by a 4-byte *home ID*, which is assigned by the controller to a device during the pairing process. All nodes paired to a given controller share the same home ID, which is assigned to the controller by the vendor in the factory.

The second form of identification is the *node ID*. A node ID is a byte value also assigned by the controller to a device during the pairing process. The controller node

always has the lowest node ID of 1. The first device paired to a controller has a node ID of 2. Subsequent pairing operations result in the assignment of an unused and monotonically increasing node ID.

Command Class.

Z-Wave networks are composed of controllers, sensors, and actuator devices. To ensure device compatibility, the application layer protocol is well-defined [Ope15b]; however, not all devices require the ability to participate in every application layer transaction. For example, a light switch does not need to know how to respond to a request for a temperature reading. Consequently, the application layer is partitioned by functionality into a series of *command classes*. Within each command class is a subset of the application layer commands pertaining to the class. A given Z-Wave device belongs to one or more command classes and the associated application layer protocol functionality is included during compilation of its firmware image. A device announces this set of supported classes to the controller during the pairing operation. Each command class is a unique byte value [Ope15b]. For example, a light switch may announce that it uses the *Binary Switch* command class, which provides commands to get, set, and report the state of the switch [Ope15b]. Given the announcement, the controller is now aware of a subset of commands that the light switch obeys. Commands sent to the light switch from unsupported command classes are ignored.

Related Work.

While there are publications regarding the reverse engineering and security assessment of other aspects of Z-Wave systems, the work herein is the first to examine security vulnerabilities of the routing protocol. In [BRMM16], the non-volatile memory components associated with a common Z-Wave transceiver chip are extracted

and analyzed. Several data structures, including the node adjacency table used for route construction by a Z-Wave controller, are identified. Building on the results from [BRMM16], the authors analyze the Z-Wave security layer in [BR16]. They discover the relationship between the AES keys and where they are stored in the EEPROM of the device. This leads to the identification of a key extraction vulnerability, where the keys are extracted from devices lacking physical security and used to send encrypted and authenticated commands to other secure devices in the network.

Other related activities include one of the earliest investigations of Z-Wave security, where a vulnerability is exploited in a Z-Wave door lock device which allows attackers to reset the Z-Wave communication key. With the key reset and known by the attacker, the door can be disengaged remotely [FG13]. In [FR15], a Z-Wave controller is exploited, using its Internet access point and web server, to add arbitrary rogue devices to a home automation network. An intrusion detection system is proposed for Z-Wave in [FRRP17] to detect outsider activities within the network. The application protocol is exploited in [HR16] to lower the lifetime of fluorescent bulbs powered through Z-Wave power switches. The susceptibility of Z-Wave networks to passive and active reconnaissance, device enumeration, and fingerprinting are explored in [BFH⁺15], [PR15], [BBTR15], and [HRRL16].

6.3 Reversing Engineering and Security Assessment Methodology

Reverse engineering and security assessment are performed using black box analysis [SGA07]. The effort consists of a series of experiments, where the network under study is exposed to stimuli while passively collecting frames emitted by the devices of the network. The captured frames are analyzed to reach conclusions that motivate further experimentation. Stimuli are generated by invoking legitimate network traffic using the devices of the network, which is especially useful for protocol reverse

engineering. Alternatively, a *SDR test platform* is used to interact with devices in the network under study at the network layer. The test platform has the capability to send and receive Z-Wave frames, allowing the edge-cases of the protocol to be examined to discover security vulnerabilities.

Network Under Study.

The network under study is a collection of four Z-Wave devices and a single controller, sharing a Z-Wave home ID of 0x018509FF. An Aeon Z-Stick2 is the Z-Wave controller for the network [Z-S16]. It runs the static controller library version 2.78 and is Node 1. Nodes 3, 4, and 5 are Aeon appliance power switches [Aeo17]. The devices report using Z-Wave library version 3 and protocol version 2.78. To avoid a network composed of devices from a single vendor and firmware version, Node 2 is a GE outdoor power switch with Z-Wave library version 6 and protocol version 3.67 [GES17]. The four devices are always powered, so they do not require *beaming* to wake them up to receive frames [ITU12a]. Each has the capability of routing Z-Wave frames and are depicted in Figure 62.

While several topologies are examined for the work herein, a single topology is used to provide a consistent demonstration platform to present the results of the reverse engineering and security analysis efforts. The demonstration topology of the network under study is shown in Figure 63. All devices in the topology are assumed to transmit at approximately the same power level. This allows the bi-directional link assumption to be made for all links of the topology [NKSK02]. The power switches are placed in a ring configuration, and the controller is only connected to Node 2. The topology is chosen to provide multiple paths between nodes and to force multihop routing between the controller and Nodes 3, 4, and 5.

To enable network-wide passive collection, the devices of the demonstration topol-



Figure 62. Devices used in the network under study. From top to bottom, this includes an Aeon USB ZStick2, a GE Outdoor Switch, and three Aeon Appliance Switches.

ogy are physically located within a single hop distance; however, this results in a fully-connected mesh topology. The demonstration topology is derived from the mesh topology using the security vulnerabilities identified in Section 6.5. Links are manually removed from the fully connected topology using Neighbor List (NL) and SR cache update messages until the topology conforms to Figure 63. Being all within transmission range, the nodes in the network under study observe shorter routes; however, they do not update their topology information with more optimal routes. As a result, the network topology remains stable.

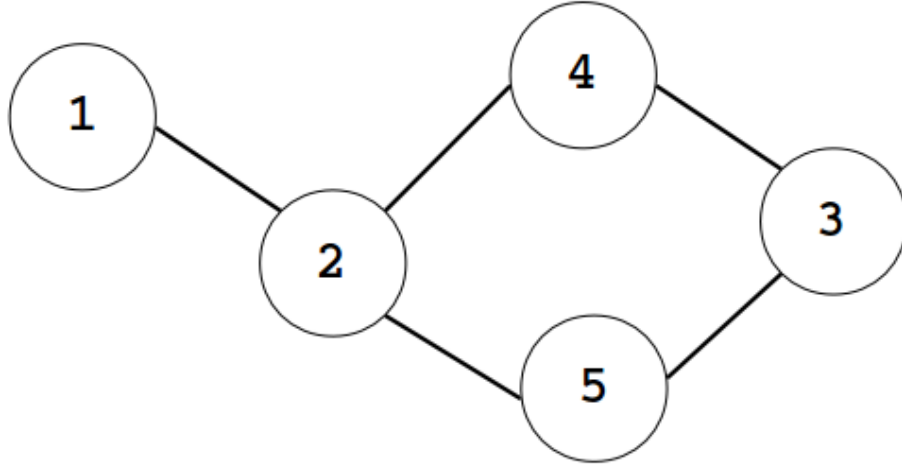


Figure 63. The topology of the network under study used for demonstrating the results of the reverse engineering and security analysis findings.

The SDR Test Platform.

From [ITU12a], there are three data rate configurations R1, R2, and R3. Each configuration provides a unique combination of symbol rate, center frequency, modulation technique, and data encoding. To date, open source implementations of the PHY layer of a Z-Wave transceiver exist for R1 and R2, but not R3. The two major open source PHY layer implementations are from Scapy-Radio [Sca16] and KillerZee [Kil16]. Scapy-Radio provides a working R2 configuration.

While KillerZee offers both R1 and R2 configurations, the frame reception rate is observed to be lower than the Scapy-Radio configuration. Upon analysis of the source code, the receive buffering mechanism may explain the low reception rate. When detecting an arriving frame, KillerZee is agnostic of the end of the frame and demodulates symbols until a fixed number of bytes are received. During the fixed-byte demodulation, a portion of another frame, such as an ACK, may also be received. While KillerZee looks for multiple frames in a receive buffer, it is unable to recover fragmented frames between fixed-byte demodulations. To account for the low ACK frame reception of KillerZee and lack of a working R1 configuration for Scapy-

Radio, both implementations are used. Empirically, the majority of Z-Wave traffic is observed at R2, with R1 traffic being observed in special situations, including pairing operations.

The *SDR test platform* refers to a PC running both R1 and R2 SDR transceiver stacks. The stacks coexist on the same host computer, provided there is at least one USB3 and two other USB interfaces available. By utilizing disjoint UDP ports, the test platform sends and receives Z-Wave frames using R1 and R2 configurations. Figure 64 provides a portrait of SDR test platform used for the research herein.

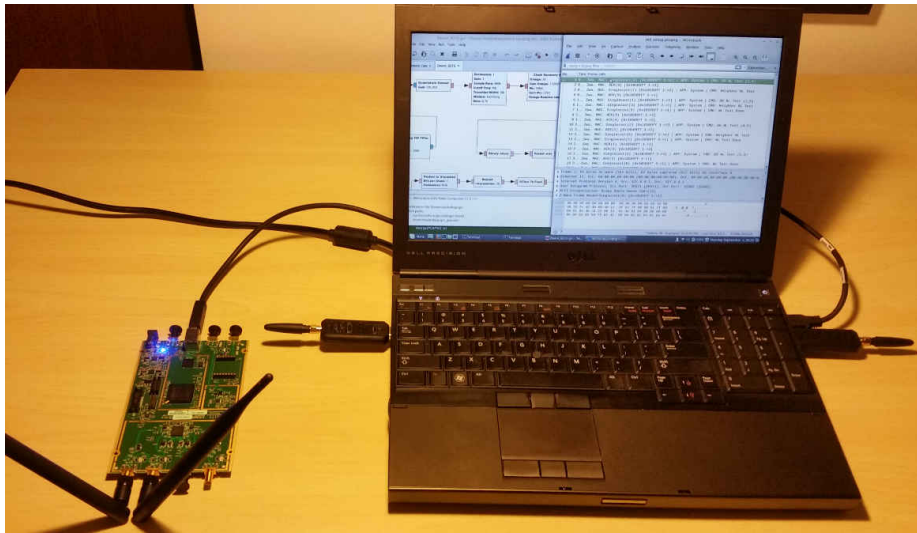


Figure 64. The SDR Test Platform: The Ettus B210 SDR is shown in the lower left. The laptop has two YARD Stick One SDRs, shown on the left and right side of the laptop.

R2 Transceiver Stack.

An architecture diagram of the R2 transceiver stack is provided in Figure 65. The Ettus B210 SDR, also known as a USRP, is used to provide a duplex interface between digitized I/Q baseband signals and analog RF signals. With regard to the receiver path, the B210 provides demodulated digital I/Q data of a filtered RF signal, received at the RX antenna, to a host computer using USB3. The USB3 data

channel is regulated by the host using a USRP Hardware Driver (UHD) device driver 003.010.000. The I/Q data is passed to a GNUradio 3.7.5 layer, which is running the R2 transceiver provided by Scapy-Radio. The Scapy-Radio transceiver filters the digital baseband samples and demodulates the samples into symbols. It also provides preamble synchronization and SOF detection for Z-Wave PPDUs at R2.

The Z-Wave MPDUs are extracted from demodulated PPDUs by the Scapy-Radio transceiver and sent over the local interface of the host computer to UDP port 52002. Any application listening on this port may receive these Z-Wave frames. For this effort, the primary recipient includes Wireshark 2.0.1 [Fou17], with a custom dissector for Z-Wave frames encapsulated in the Scapy-Radio header. In addition, a custom shared library *libZwave.so* is developed so arbitrary applications may send, receive, and process frames as active participants of a Z-Wave network.

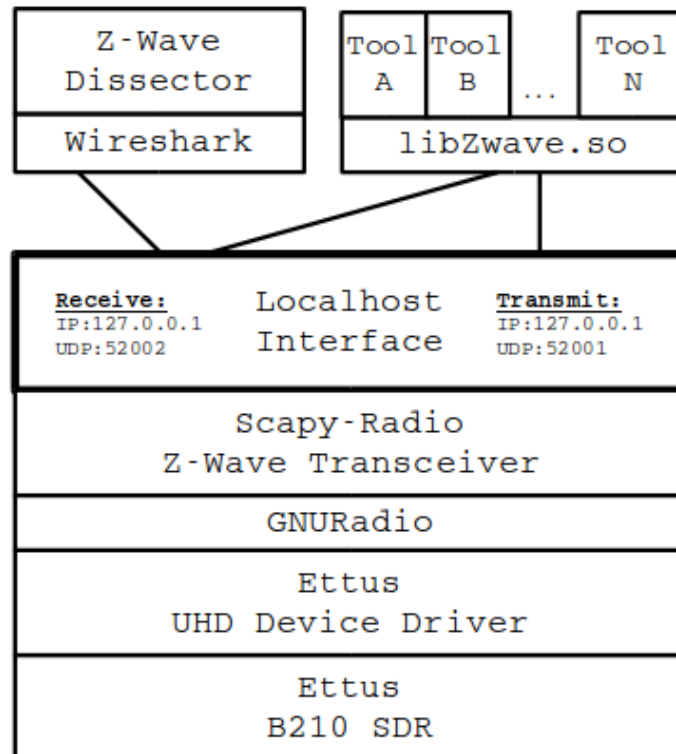


Figure 65. Protocol stack for SDR-based Z-Wave R2 transceiver.

The transmit path is similar to the receive path, but in the opposite direction. Applications may transmit a Z-Wave MPDU by encapsulating it with a Scapy-Radio header and sending it over the local interface to UDP port 52001. The Scapy-Radio demodulator listens for arriving datagrams on this port and appends the PPDU header, consisting of a preamble and SOF, modulates the frame at baseband, and sends the digitized signal to the B210 SDR. The SDR upconverts the digital I/Q data to an analog RF signal and emits it through the TX antenna.

R1 Transceiver Stack.

An architecture diagram of the R1 transceiver stack is provided in Figure 66. The SDR for R1 traffic is the YARD Stick One. This SDR provides simplex communication, so two are required to provide duplex communications. The modulation and demodulation occurs on the SDR. RFCat provides data and control channels to the SDR. KillerZee 0.1 uses RFCat 1.0 to configure the YARD Stick One to send or receive Z-Wave frames at either R1 or R2. KillerZee also has an interface to the data channel of the radio using RFCat.

Two Python scripts are added to the R1 stack to make it compatible with the R2 stack. ZwDump is a tool provided in KillerZee, which is extended to take frames received over the radio, append a Scapy-Radio header, and send the frame over the local interface to UDP port 52004. ZwPlay is custom script added to KillerZee to listen on UDP port 52003. A frame received on this port is stripped of its Scapy-Radio header and sent to the SDR for transmission using the KillerZee interface. This scheme allows Wireshark and libZwave.so to send or receive R1 traffic without modification.

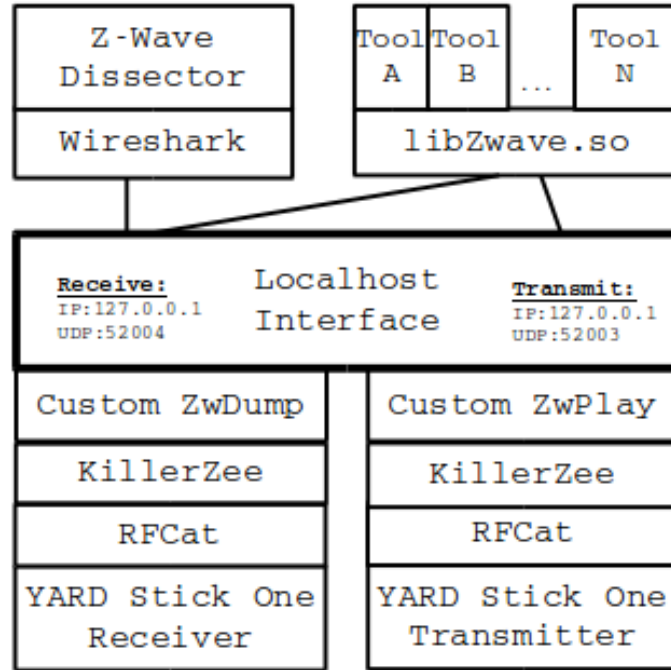


Figure 66. Protocol stack for SDR-based Z-Wave R1 transceiver.

Collection and Injection Techniques.

The SDR Test platform is used for passive collection and frame injection. For the demonstration topology, the SDR test platform is able to observe transmission of the entire network under study by exploiting the security vulnerabilities identified in Section 6.5. In addition to passive collection, the test platform is used to inject stimuli into the network under study. Tools are derived to allow byte values of Z-Wave frames to be manually constructed and injected into the network. Routed frames are manually constructed, injected into the network, and observed while being forwarded over the network under study. The observations lead to the identification of data field semantics, which are confirmed by additional frame injection activities.

In addition to simple frame injection, some aspects of the protocol are found to require more complex interaction. For example, to participate in frame forwarding, the test platform must receive, parse, update, and transmit the frame. The libZwave.so

library implements a significant portion of the MAC and security layers. As routing details are discovered, these are also added to the library to facilitate more complex interactions between the SDR test platform and the network under study. Unless otherwise stated, the SDR test platform uses the node IDs of 10 and 11 in the source field of injected frames to be distinguishable from the legitimate nodes in the network.

Reverse Engineering Process.

A direct approach to reverse engineering is to extract and statically analyze firmware, memory, and other non-volatile artifacts [TETH16, MLM17]. While it is possible to extract the firmware of Z-Wave devices, the reverse engineering challenges identified in [BRMM16], such as the proprietary special function registers, make it challenging to isolate the routing functionality using static analysis. A network protocol is inherently extrinsic, so the semantics may be derived by collecting and analyzing protocol messages [MPAM15]. In general, the forwarding and topology management aspects of the Z-Wave routing protocol are reverse engineered using a manual black-box analysis process and protocol fuzzing [SGA07], guided by available open-source literature such as [Pae13] and [BRMM16]. The following steps are taken:

1. Normal protocol traffic is collected from the network under study.
2. Network traces are analyzed to identify differences in frames to isolate network protocol fields.
3. The semantics of the isolated fields are identified by injecting crafted frames into the network and observing the response. Through trial and error, the behavior of a field emerges. The semantics are confirmed by observing the network react as predicted to several variations of a given field.

4. Unknown fields are fuzzed, where frames are injected with arbitrary field values to discover semantics. Cases where a field's purpose remains hidden are otherwise noted herein.
5. Having identified a sufficient number of fields, the reverse engineering process focuses on the meaning of the messages containing the fields. Again, the SDR test platform injects targeted network messages into the network under study. The responses are observed and analyzed to estimate semantics. The semantics are confirmed by injecting similar frames into the network and observing predictable behavior.
6. Protocol transactions are realized as a composition of known network messages. The meaning of transactions are discovered and confirmed using the same techniques performed for fields and messages.

Security Analysis Process.

Through reverse engineering, a model of the forwarding and topology management aspects of the Z-Wave routing protocol is realized. The security state of the model is evaluated *by inspection* [AY07] for its susceptibility to classic integrity-based routing attacks performed by an outsider node. Device impersonation, manipulation of forwarded frames, and topology state corruption are all explored using the SDR test platform and network under study. Assuming the Z-Wave protocol is resistant to the identified integrity-based routing attacks, the goal of the security analysis activity is to discover violations to the assumptions. Similar to the reverse engineering process described in Section 6.3, violations are discovered interactively through an iterative process of injecting a given stimulus into the network, observing the network response, and analyzing the results to guide the subsequent experiments.

6.4 The Reverse Engineered Z-Wave Routing Protocol

The results of the reverse engineering analysis on the protocol are provided in the following sections. First, the forwarding mechanisms are presented. This is followed by details of topology and route management, including the data structures and protocol coordination messages.

Source Routing.

Routed Z-Wave frames use a network header, located between the MPDU header and application layer. Figure 67 shows the fields of the network header. The first two bytes of the network header are divided into four 4-bit nibbles. The first nibble is the *failed hop*, used only for route error messages to declare the hop where the error occurred and is otherwise zero for other types of routed frames. The second nibble holds the SR type, which is used by routing nodes to determine how to forward the SR. The third nibble holds the length of SR in bytes. The fourth nibble holds the hop index field, which maintains the state of the SR while it is forwarded. The remaining bytes are the SR. The i^{th} byte in the SR is the node ID of the i^{th} hop in the route. The SR only contains the inner nodes of the route, relying on the MPDU header to provide the node IDs of the route endpoints. The SR is limited to four inner node hops [Pae13]. Considering the implicit final hop to the destination, Z-Wave routes may be up to five hops in length.

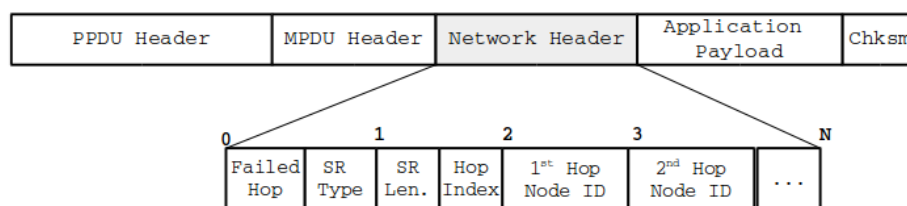


Figure 67. The Z-Wave Network Header format.

Forwarding Behavior.

As with DSR, the forwarding behavior of the inner nodes of a route is simple [JHM07]. Since the routing layer is optional, the state of the *routed* flag in the control field of the MPDU header is used to resolve the presence of the network header [ITU12a]. Upon receipt of a frame with this bit set, a node determines if it is responsible for forwarding the frame. The hop index field in the network frame provides the byte offset in the SR of the next hop. If a node's ID is located at this position, then the node updates the hop index field according to the route type field, recalculates the frame checksum, and retransmits the frame.

Table 6 summarizes which node is responsible for forwarding a routed frame as a function of the hop index, where N is the SR length. Since hop index is a byte offset, it is zero-based rather than a hop count. The destination realizes it is supposed to receive the frame when the hop index value matches the SR length field. While routed frames are normally forwarded to the destination, certain types of routing frames are reverse-routed to the source node. The source is able to recognize that it is the intended recipient when the hop index is 0xf. Since the hop index field is a 4-bit value, the field incurs an underflow when the node at hop index zero decrements the hop index before forwarding to the source.

Table 6. Forwarding Targets for a SR of length N , $1 \leq N \leq 4$.

Hop Index Value	Next Hop Target
0 to $N - 1$	In the SR, indexed by the hop index
N	Destination node in MPDU header
0xf	Source node in MPDU header

Routing Frame Types.

Three types of routing frames are summarized in Table 7. Application frames are routed from a source to destination using routing frame type 0x00. The other

two routing frame types are used to inform the source node of the state of a routed application frame. A route ACK is issued by the destination node to confirm a given routed application frame is received. A route No-Acknowledge (NACK) is issued by an inner node when a forwarding error is detected. The routing behavior for each type is described in the following subsections.

Table 7. Known Types of Routing Frames.

SR Type Value	Description	Hop Index Behavior
0x00	Application Frame	Increments
0x03	Route ACK	Decrements
0x05	Route NACK	Decrements

Routed Application Frame.

When a source node needs to send a message to a target that is not a one-hop neighbor, it creates a routed application frame. The destination field of the MPDU header is set to the target of the frame and the routed bit is set. A network header is inserted between the MPDU header and application payload. The source node selects a route from a SR cache, embeds the SR into the network header, and sets the route length field to the length of the selected route. The route type field is set to 0x00 to designate it as a routed application frame. The hop index field is set to zero to designate the next hop recipient as the first node identified in the SR. Finally, the source transmits the application frame. The first hop node identifies itself as responsible for forwarding the frame. Before retransmitting the frame, the hop index field is incremented to designate the node listed at the one byte offset of the SR as the next hop. The source and destination fields of the MPDU are not updated while the frame is forwarded. The destination realizes it is the recipient when it receives a routed application frame where the hop index is equal to the route length.

Route Acknowledgement.

A source node receives confirmation that a routed frame arrives at a destination node when it receives a route acknowledgement. Similar to MAC layer ACKs, the sender makes up to three attempts to send a routed application frame before giving up. Retries are conducted after failing to receive a route ACK within a certain time interval.

Upon receipt of a routed frame, the destination node prepares a route ACK. The route ACK MPDU header and network header are copied from the received application frame. The source and destination fields of the MPDU are swapped. The SR type field is set to 0x03 to designate it as a route ACK and the hop index is decremented before transmitting the route ACK. Without having to reverse the ordering of the SR in the header, the message traverses the reverse path of the SR by having each hop decrement the hop index field before retransmitting the frame. The source receives the route ACK when the hop index field is 0xf and uses the sequence number field in the MPDU header of the route ACK to identify the routed application frame being confirmed.

A routed application frame and associated route ACK are demonstrated on the network under study. The controller is invoked to toggle the switch state of Node 3 to ON. Figure 68 shows the observed traffic. Frames 1-3 show the command message being routed from the controller to Node 3, where the hop index is incremented at each hop. Upon receipt of the command message, Node 3 sends a route ACK back to the controller. Frames 4-7 show the hop index decrementing as it is retransmitted, arriving back at the controller at frame 6.

To provide further clarity, Figure 69 depicts the observed frames as a protocol activity timeline. The nodes along the SR are listed at the top. Each arrow corresponds to an observed frame in Figure 68. The state of the source route is included

Frame Number	Home ID	Source Node ID	Destination Node ID	Route Type	Command Class
No.	Time	Info			
1	0.0000...	MAC: Singlecast(5)	[0x18509ff 1->3]	NET: SR [2,5]@0	APP: Switch
2	0.0127...	MAC: Singlecast(5)	[0x18509ff 1->3]	NET: SR [2,5]@1	APP: Switch
3	0.0209...	MAC: Singlecast(5)	[0x18509ff 1->3]	NET: SR [2,5]@2	APP: Switch
4	0.0286...	MAC: Singlecast(5)	[0x18509ff 3->1]	NET: ACK [2,5]@1	
5	0.0358...	MAC: Singlecast(5)	[0x18509ff 3->1]	NET: ACK [2,5]@0	
6	0.0485...	MAC: Singlecast(5)	[0x18509ff 3->1]	NET: ACK [2,5]@f	

Elapsed Time (Seconds) Frame Type Sequence Number First Hop Node ID Second Hop Node ID Hop Index

Figure 68. The frames observed in the network under study when a SR is routed from Node 1 to 3 and the corresponding route ACK taking the reverse route back to Node 1.

below each frame. The figure shows the application frame being routed from Node 1 to 3. After receiving the routed application frame, Node 3 replies with a route ACK back to Node 1.

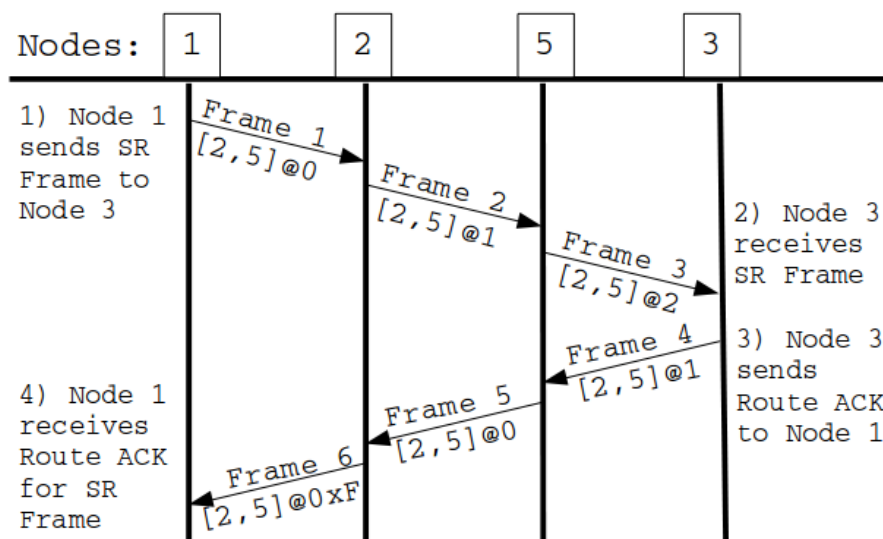


Figure 69. A protocol activity diagram of a SR routed from Node 1 to 3 and the route ACK taking the reverse route back to Node 1.

Route Error.

Nodes forwarding an application frame are responsible for detecting and reporting observed routing errors to the source node. After forwarding a routed frame, a node awaits confirmation that the frame is received by the next hop. In all cases except

for the last hop, confirmation is made by witnessing the next hop appropriately forwarding the message to its next hop. Since the last inner hop node does not observe forwarding beyond the destination, it sets the *acknowledgement required* flag in the MPDU header before forwarding the frame to the destination. Receipt, in this case, is confirmed by receiving a MAC layer ACK from the destination. If no confirmation is observed in a given time interval, the node assumes the next hop failed to receive the frame and attempts to resend. After at least three attempts, the node gives up and reports a routing error.

The node discovering a routing error uses a route NACK message to notify the source node. The route NACK is similar to the route ACK because it is routed in reverse of the SR in the network header. The route NACK is composed of the MPDU and network headers of the offending frame, where the source and destination fields are swapped. The SR type field is set to 0x5 to designate it as a route NACK. The hop index of node failing to confirm its receipt is copied to the failed hop field, the hop index field is decremented, and the frame is sent to begin traversal back to the source node. Upon receipt, the source node may use the failed hop field to select a new route from its SR cache that avoids the failed hop. As with a route ACK, the sequence number field in the MPDU header of the route NACK is used to associate the error with a recently transmitted frame.

To demonstrate a route NACK, the SDR test platform injects a single routed frame into the network under study. The source and destination fields are set to unused node IDs to force a routing error when the last hop of the route attempts to forward to the destination node. Figure 70 shows the results of the frame capture during the route error event. In frames 1-5, the frame is routed over the inner nodes of the SR. Since the destination is fictitious, the last inner node never receives a MAC layer ACK of its forwarded frame. After three failed forwarding attempts, shown at

frames 5-7, Node 5 generates a route NACK that is routed back to the source node in frames 8-11. The NACK in the figure identifies the fourth hop as the failed node, which is the destination node. A protocol activity time line of the observed frames is provided in Figure 71.

No.	Time	Protocol	Info
1	0.00...	Zwave	MAC: Singlecast(0) [0x18509fff 11->10] NET: SR [2,4,3,5]@0 APP: Hello / NOOP
2	0.01...	Zwave	MAC: Singlecast(0) [0x18509fff 11->10] NET: SR [2,4,3,5]@1 APP: Hello / NOOP
3	0.02...	Zwave	MAC: Singlecast(0) [0x18509fff 11->10] NET: SR [2,4,3,5]@2 APP: Hello / NOOP
4	0.13...	Zwave	MAC: Singlecast(0) [0x18509fff 11->10] NET: SR [2,4,3,5]@3 APP: Hello / NOOP
5	0.13...	Zwave	MAC: Singlecast(0) [0x18509fff 11->10] NET: SR [2,4,3,5]@4 APP: Hello / NOOP
6	0.18...	Zwave	MAC: Singlecast(0) [0x18509fff 11->10] NET: SR [2,4,3,5]@4 APP: Hello / NOOP
7	0.27...	Zwave	MAC: Singlecast(0) [0x18509fff 11->10] NET: SR [2,4,3,5]@4 APP: Hello / NOOP
8	0.29...	Zwave	MAC: Singlecast(0) [0x18509fff 10->11] NET: NACK@4 [2,4,3,5]@2
9	0.30...	Zwave	MAC: Singlecast(0) [0x18509fff 10->11] NET: NACK@4 [2,4,3,5]@1
10	0.31...	Zwave	MAC: Singlecast(0) [0x18509fff 10->11] NET: NACK@4 [2,4,3,5]@0
11	0.32...	Zwave	MAC: Singlecast(0) [0x18509fff 10->11] NET: NACK@4 [2,4,3,5]@f

Failed Hop Field

Figure 70. Captured frames as the result of the SDR platform injecting a SR frame into the network under study using imaginary source and destination IDs. Because Node 10 does not exist, Node 5 fails to forward the frame and generates a route NACK that takes the reverse route back to the source. The NACK frame indicates the fourth hop failed.

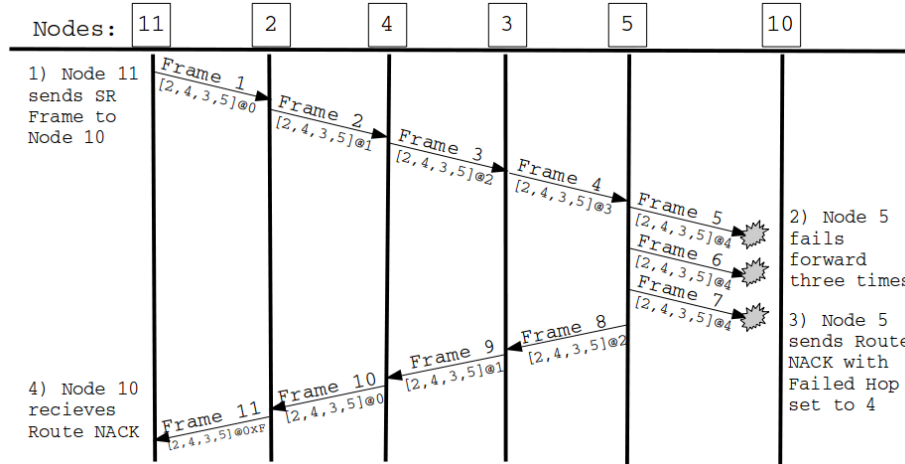


Figure 71. A protocol activity diagram of a SR routing from Node 11 to 10. Because Node 10 does not exist, Node 5 is unable to successfully forward the frame. It generates a route NACK, which takes the reverse route back to source Node 11.

Network Management.

Figure 72 depicts the data structures and activities necessary for topology maintenance and route discovery. Local topology information resides in each routing node

as a one-hop NL. The global topology state resides in the adjacency table maintained by the controller. The adjacency table is updated by polling each node for its NL [Pae13]. Each node also stores several caches of SRs. New routes are generated on demand by the controller, where a routing node makes a request for a SR to a given target. Having the global adjacency table, the controller is able to realize multiple routing solutions and provide them to the requesting node, where they are cached.

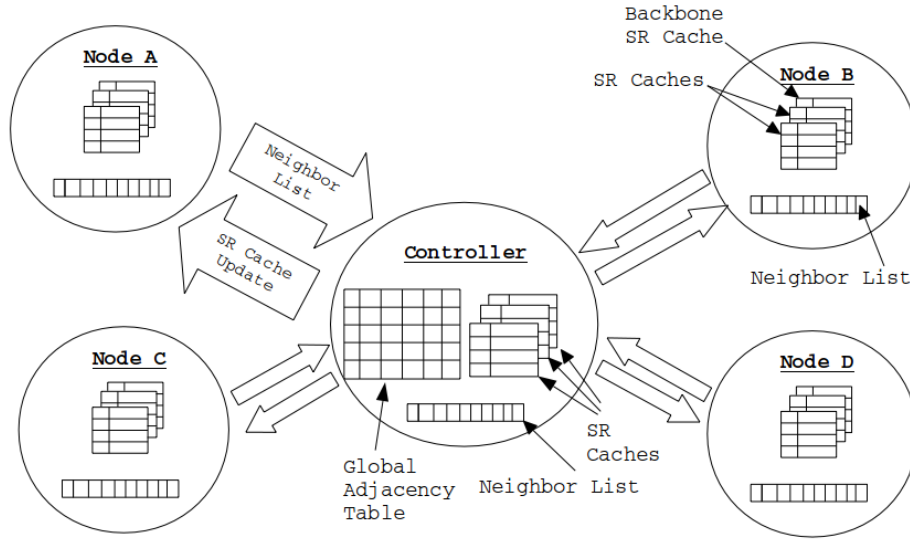


Figure 72. The Z-Wave Network Management Architecture.

In addition to a NL and several SR caches, routing nodes also have a backbone SR cache to be able to reach the controller. OpenZwave defines a route from the node to the controller as a *reverse route*; herein, it is designated as a *backbone route* to more appropriately describe its criticality. A node oblivious of backbone routes is unable to request new routes from the controller.

From a holistic perspective, the routing protocol has properties found in both reactive and proactive routing protocols. Like the proactive routing protocol Optimized Link Source Routing (OLSR), the Z-Wave routing protocol collects local topology state information from each node before any routes are needed. Like the reactive routing protocol DSR, routes are discovered at the request of the source

node; however, the discovery is performed by the controller using its global topology state. Having aspects of both reactive and proactive routing protocols, Z-Wave can be considered a hybrid routing protocol. The following subsections describe the observed data structures, coordination messages, and transactions involved in maintaining topology state.

Coordination Messages.

Several coordination messages are identified as being associated with topology management and route discovery, which are summarized in Table 8. Note that all of the coordination messages belong to command class 0x01. This command class provides commands outside of the scope of network routing, including pairing operations, network and node ID assignments, and interactions between primary and secondary controllers. Open source literature such as [Ope15b] do not name this command class, so it is designated herein as the *System* command class.

Table 8. Known network commands of the System command class.

Command	Description	Parameters
0x04	Do NL Test	NL Primitive
0x05	Get NL	Target Node ID
0x06	Report NL	NL Primitive
0x07	NL Test Done	None
0x0C	SR Cache Assignment	SR Cache Entry Primitive
0x14	Backbone Cache Assignment	SR Cache Entry Primitive
0x15	SR Request	Target Node ID
0x18	NL Test	Unknown Parameter

Several coordination messages utilize one of two identified *primitive* data structure formats. The first data structure is the NL primitive. It is used to convey topology state within topology management coordination messages, and its basic structure is shown in Figure 73. The first byte of the primitive indicates the length of the remaining bytes. The length field is followed by a variable length bitfield.

The last byte of the primitive has an impact on route selection; however, only two values are observed. When this byte is 0x10, the route is used by the recipient node. When the byte is 0x08, some devices avoid using the route entirely. Other devices give an entry with a status of 0x08 a lower priority, using it only after routes marked with a status of 0x10 are exhausted. While the behavior is consistent with a status or priority indicating field, the true meaning of this field may not be discovered until the firmware is fully reverse engineered.

NL Updates.

A node updates its NL based on the results of NL tests, initiated upon the receipt of a **Do NL Test** message. The payload of this message contains a NL primitive, which describes how the tests are to be conducted. A set bit in the NL primitive instructs the node to perform a NL test on the corresponding node ID. When the node encounters a bit that is not set, this tells the node to remove that node ID from its NL. Thus, messages with a NL primitive of a string of zero bits that covers all node IDs in the network clears the node's NL. Alternatively, a string of set bits covering all node IDs of the network forces the recipient to perform a NL test on every node ID.

A node conducts a NL test on a given target using a **Do NL Test** message. The node sends this message to the target using rate configuration R1, regardless of the *de facto* PHY rate configuration. The NL test passes if the target replies with an ACK. The test fails if, after three attempts, no ACK is received. The test message has a single byte as a parameter; however, its meaning is not yet known. Once a node has performed a NL test on every requested node, it sends a **NL Test Done** to the sender of the **Do NL Test** message.

Adjacency Table Updates.

The adjacency table is updated when a controller queries a node for its NL. The query is performed by sending a **Get NL** message to a target. The target replies with a **Report NL** message, which contains the node's NL. The format of the NL primitive is conveniently consistent with the adjacency table [BRMM16]. Updating the adjacency table is as simple as performing a direct memory copy of the NL primitive to the memory location holding the adjacency record of the sender.

The controller adjacency table is updated when the device is rebooted or in response to a user invoked request to repair the network. Unsolicited **Report NL** messages are ignored by the controller of the network under study to prevent external manipulation of the adjacency table structure.

Route Request.

A node may request a route to a target using the **SR Request** message, where the destination node ID of the route is the only parameter. The response to this request depends on the capabilities of the recipient. A controller replies with a sequence of **SR Cache Assignment** messages in *cache entry index* order. The routes also appear to be in shortest-length-first order. The controller may also follow these messages with a set of **Backbone Cache Assignment** messages, to freshen the node's cache of critical routes to the controller.

If a **SR Request** is received by a routing node that is not a controller, the node replies to the sender with its backbone cache as a sequence of **Backbone Cache Assignment** messages. Having an updated backbone cache, the requesting node may now query routes from the controller. The behavior provides a recovery mechanism for nodes with stale or corrupted backbone SR caches.

The **SR Request** message is discovered through protocol fuzzing of command byte

values for the *System* command class. While the SDR test platform can cause a node to facilitate a route request, the activity is not observed during normal operations of the network under study. As a result, the conditions necessary for a legitimate node to invoke a route request are not yet known.

A node is capable of caching multiple routes for each target. By injecting **SR Request** messages into the network under study, observations of the resulting cache assignment messages have never exceeded four entries per target. Without dynamic and static analysis of the device firmware, further aspects of the cache remain hidden.

Route Selection.

A node consults its SR cache when it requires a multihop route. When more than one solution is available, the source node must make a selection. If the selected route fails, the source node must select an alternative route. The selection process is repeated until the cache solutions are exhausted or the source node gives up. The route selection behavior is studied by isolating a node from the network under study and causing it to attempt to route an application frame to a target. As the source node incurs routing failures, it selects alternative routes, and eventually concedes. The observations are compared with the node's SR cache for the given target to reveal the route selection behavior.

For the experiment, Node 3 is the isolated node that attempts to send a routed application frame to Node 2. Figure 75 shows three cache assignment messages sent to Node 3 prior to the injected frame. The first cache entry holds a path to Node 2 through Node 5. The second cache entry holds a route to Node 2 through Node 4. The third entry is empty and marked invalid.

Figure 76 shows the reaction of Node 3 in response to the injected frame and after the other nodes are disabled. The first frame in the capture is the frame sent by the

No.	Time	Info
1	0.00...	MAC: Singlecast(0) [0x18509ff 1->3] APP: System(0x0c) CMD: SET SR Cache 2.0:[5] Valid
2	0.00...	MAC: ACK(0) [0x18509ff 3->1]
3	2.00...	MAC: Singlecast(0) [0x18509ff 1->3] APP: System(0x0c) CMD: SET SR Cache 2.1:[4] Valid
4	2.00...	MAC: ACK(0) [0x18509ff 3->1]
5	4.00...	MAC: Singlecast(0) [0x18509ff 1->3] APP: System(0x0c) CMD: SET SR Cache 2.2:[Empty] Invalid

Source Route Entry Entry Status
 Target Node ID Cache Index

Figure 75. SR cache assignments for Node 3.

SDR test platform, which spoofs as Node 2 asking Node 3 for its state. After replying with a route ACK, Node 3 attempts to send the application layer response through Node 5 in frames 5-7, which is the first cache entry for Node 2. After not receiving a route ACK from Node 2, it makes three attempts to route through Node 4, which is the second cache entry for that target. After this fails, it gives up.

No.	Time	Info
1	0.000...	MAC: Singlecast(0) [0x18509ff 2->3] NET: SR [10,11]@2 APP: Swi
2	0.007...	MAC: Singlecast(0) [0x18509ff 3->2] NET: ACK [10,11]@1
3	0.035...	MAC: Singlecast(0) [0x18509ff 3->2] NET: ACK [10,11]@1
4	0.081...	MAC: Singlecast(0) [0x18509ff 3->2] NET: ACK [10,11]@1
5	0.104...	MAC: Singlecast(1) [0x18509ff 3->2] NET: SR [5]@0 APP: Switch
6	0.153...	MAC: Singlecast(1) [0x18509ff 3->2] NET: SR [5]@0 APP: Switch
7	0.222...	MAC: Singlecast(1) [0x18509ff 3->2] NET: SR [5]@0 APP: Switch
8	0.247...	MAC: Singlecast(2) [0x18509ff 3->2] NET: SR [4]@0 APP: Switch
9	0.316...	MAC: Singlecast(2) [0x18509ff 3->2] NET: SR [4]@0 APP: Switch
...	0.403...	MAC: Singlecast(2) [0x18509ff 3->2] NET: SR [4]@0 APP: Switch

Figure 76. Route selections for Node 3 (Aeon Appliance Switch).

The experiment is repeated; however, Node 2 becomes the isolated node attempting to send an application frame to Node 3 over the network. The SR cache in Node 2 for Node 3 is identical to the SR cache in Node 3 for Node 2, shown in Figure 75. With all nodes disabled except for Node 2, the SDR test platform injects the same frame and the observed behavior is shown in Figure 77. In frame 2, Node 2 initially selects the reverse route of the received frame as the SR of the reply message. After the reverse route fails once, it uses its first and second cache entries. After they fail, it makes several more attempts to use the reverse route of the request in frames 9-11. The last three frames of the figure show Node 2 attempting to reach Node 3 locally,

where a SR is not used. Although both nodes have the same route entries in their SR cache, the route selection behavior for Node 2 is different from Node 3. The difference may be explained because Node 2, the GE switch, has a newer library and protocol version than Node 3, as reported in Section 6.3.

No.	Time	Proto	Info
1	0.0000...	Zwa...	MAC: Singlecast(0) [0x18509fff 3->2] NET: SR [10,11]@2 APP: Swit
2	0.1788...	Zwa...	MAC: Singlecast(12) [0x18509fff 2->3] NET: SR [11,10]@0 APP: Swi
3	0.2810...	Zwa...	MAC: Singlecast(13) [0x18509fff 2->3] NET: SR [5]@0 APP: Switch
4	0.4241...	Zwa...	MAC: Singlecast(13) [0x18509fff 2->3] NET: SR [5]@0 APP: Switch
5	0.4548...	Zwa...	MAC: Singlecast(14) [0x18509fff 2->3] NET: SR [4]@0 APP: Switch
6	0.5367...	Zwa...	MAC: Singlecast(14) [0x18509fff 2->3] NET: SR [4]@0 APP: Switch
7	0.5368...	Zwa...	MAC: Singlecast(14) [0x18509fff 2->3] NET: SR [4]@0 APP: Switch
8	0.5595...	Zwa...	MAC: Singlecast(14) [0x18509fff 2->3] NET: SR [4]@0 APP: Switch
9	0.5905...	Zwa...	MAC: Singlecast(15) [0x18509fff 2->3] NET: SR [11,10]@0 APP: Swi
10	0.6822...	Zwa...	MAC: Singlecast(15) [0x18509fff 2->3] NET: SR [11,10]@0 APP: Swi
11	0.7332...	Zwa...	MAC: Singlecast(15) [0x18509fff 2->3] NET: SR [11,10]@0 APP: Swi
12	0.7614...	Zwa...	MAC: Singlecast(15) [0x18509fff 2->3] APP: Switch Binary
13	0.7641...	Zwa...	MAC: Singlecast(15) [0x18509fff 2->3] APP: Switch Binary
14	0.9070...	Zwa...	MAC: Singlecast(15) [0x18509fff 2->3] APP: Switch Binary
15	0.9127...	Zwa...	MAC: Singlecast(15) [0x18509fff 2->3] APP: Switch Binary

Figure 77. Route selections for Node 2 (GE Outdoor Switch).

6.5 Security Analysis on Z-Wave Routing Protocol

In this section, the integrity and reliability implications of the Z-Wave routing protocol are analyzed. Several vulnerabilities are discovered on the network under study using the SDR test platform.

Existing Security in Z-Wave.

Z-Wave utilizes several security mechanisms, which include cryptographic, policy-driven, behavior detection, and out of band mechanisms. They are summarized in the following subsections.

Security Command Class.

Z-Wave provides a confidentiality, source integrity, and data integrity service through the *Security* command class. Application frames may be encapsulated in

a security frame that is both encrypted and signed. The frame is secured through symmetric encryption using AES and three shared keys, known by every node of the network requiring the security service. The operations of the *Security* command class are described in detail in [FG13] and [BR16].

During network inclusion, the device specifies which of its supported command classes must use the security layer. When an application layer command is exchanged for one of the designated command classes, it must be embedded within a secure frame. Otherwise, the frame is ignored by the device [BR16].

Empirical evidence shows that the utilization of the security service is proportional to the purpose of the device. For example, physical security related devices such as door locks, motion detectors, and alarms are more likely to use the security service than smart energy devices such as light bulbs, appliance switches, and thermostats.

The *Security* command class has particular limitations when used. First, only the application layer message is encrypted. The fields in the MPDU header, network layer, and security frame header are not encrypted. Data integrity is provided through signature checking, which only applies to the fields being signed. In addition to the application payload, this includes the MPDU source node ID, destination node ID, and MPDU length field [BR16]. Note that none of the network header fields are involved in the signature, so these fields may be modified in transit without causing the signature check to fail. Since all paired devices use the same authentication key, source integrity at the node ID resolution is not possible. Instead, devices may only discriminate between messages originating from trusted and untrusted sources.

While none of the devices in the network under study utilize the *Security* command class, the Yale Z-Wave door lock analyzed in [BR16] possesses this capability. While the door lock silently drops door unlock commands sent by the SDR test platform, it responds to *System* command class messages. The implication is that the *System*

command class is not required to use the security layer.

Out of Band Triggers.

Many of the critical activities lack a way to be initiated over the network. Instead, they require physical proximity and human intervention for invocation. For example, pairing operations require the user to reset a device, press a button, or reinsert batteries to force a node into pairing mode. Another example of an out of band protection involves updates to the adjacency table in the controller. Since unsolicited NL reports are ignored, an attacker attempting to corrupt the topology state must react to a request from the controller. Known ways of causing a controller to update its adjacency table include rebooting the device or requesting a network healing action through the controller’s user interface. Ways of forcing these events through the Z-Wave network remain unknown; however, network healing may be susceptible to the rogue controller vulnerability found in [FR15]. If Z-Wave devices are powered through Z-Wave switches, the switches may be remotely toggled to induce a reset event. While the Z-Wave serial API has the ability to reset a Z-Wave transceiver [Ope15a, BRMM16], an attack vector to utilize the API through the Z-Wave network has not yet been discovered.

Privileged Controller.

The controller is privileged because it possesses both global topology information and node capability information for each node that is paired to the network. A controller may identify and react to integrity-based attacks by correlating events with its global knowledge. Conversely, non-controller nodes are less secure because they do not have access to global information necessary to identify inconsistencies. For example, the controller ignores messages sent from the SDR test platform if the

source ID is of a node that is never paired to the network. The other nodes in the network under study are unable to recognize outsider nodes. As a result, they respond to unsolicited frames from arbitrary sources so long as the source node ID is in a valid range (i.e., 1 to 232) [ITU12a].

Watchdog Routing.

Route error messages are generated by a forwarding node when it fails to observe the next hop forwarding the message within a time interval. The node observing the forwarding node is known as a *watchdog*. Having each previous hop monitor the protocol compliance of the next hop assists in the identification of malicious or non-cooperative forwarding nodes within the network [HHF⁺04, BS09, TACC09]. The capabilities of watchdog routing for Z-Wave are examined in Section 6.5.

Vulnerabilities.

The Z-Wave protocol, as observed in the network under study, is analyzed for security weaknesses in data and source integrity. The results are described in the following subsections.

Impersonation.

Impersonation attacks violate the source integrity of the protocol. With the exception of the controller, Z-Wave devices implicitly trust the source and destination fields of the MPDU frame. This makes it trivial to impersonate frames originating from the controller or another device. To demonstrate impersonation, a single frame is injected into the network using the SDR test platform. The injected frame is a request; observing a response to the forged request implies that the replying node accepts the message as legitimate.

The injected frame is a **Get NL** message to Node 2. The source ID field in the MPDU header is also set to Node 2 to make it clear that the message originates from an artificial source. The assumption is that a node will not have to use its radio to query and respond to itself for data residing in its memory. The frame is emitted into the network and the responding frames are observed in Figure 78.

No.	Time	Info
1	0.0000...	MAC: Singlecast(0) [0x18509ff 2->2] APP: System CMD: GET NL
2	0.0102...	MAC: ACK(0) [0x18509ff 2->2]
3	0.0215...	MAC: Singlecast(10) [0x18509ff 2->2] APP: System CMD: REPORT NL (1,4,5)
4	0.0945...	MAC: Singlecast(10) [0x18509ff 2->2] APP: System CMD: REPORT NL (1,4,5)
5	0.1662...	MAC: Singlecast(10) [0x18509ff 2->2] APP: System CMD: REPORT NL (1,4,5)

Figure 78. SDR impersonates Node 2 by requesting the NL of Node 2, to which Node 2 replies.

The injected frame is shown as frame 1. While Node 2 acknowledges the forged frame, this only means that the MPDU header and checksum checks are valid. Proof of the impersonation vulnerability is in the remaining frames, where Node 2 attempts to tell itself its NL in response to the unsolicited request. An interesting observation is that Node 2 acknowledges frame 1 but does not acknowledge frames 3, 4, and 5. Although each of these frames has Node 2 as the destination, the node is able to avoid acknowledging frames it actually sends. This capability may be extended to identify Sybil attacks occurring on the network, where a node alerts the controller when it receives a message from itself that it did not send.

Devices using the Z-Wave security layer have some protection against outsider impersonation. Devices specify, at a command class granularity, which command messages must use the secure frame. Secure frames are signed and encrypted using keys exchanged during network inclusion. An outsider who is not in possession of the authentication and encryption keys is unable to transmit a valid secure frame. Regardless of the chosen source ID, the outsider is unable to impersonate the origin of a command message if the destination requires that it is sent in a secure frame. However, the outsider may still perform impersonation attacks on the device using

commands from a supported command class that is not required by the device to use the security layer.

Arbitrary NL Modification.

Routing nodes, including the controller, are vulnerable to having their NLs manipulated by an external source. This is shown using the SDR test platform in the network under study. Acting as outsider Node 10, the SDR test platform interacts with Node 3 to add itself to the NL of Node 3. This is performed in two steps. First, a process is executed on the SDR test platform to listen and acknowledge received frames at R1 sent to Node 10. Second, a separate process on the laptop sends Node 3 a Do NL Test message with Node 10 set as the target. This initiates the NL add activity, which is captured using Wireshark. Figure 79 shows the combined results of the activity at R1 and R2.

			Neighbor List	
No.	Time	Info		
1	0.0000...	MAC: Singlecast(0) [0x18509ff 10->3] APP: System CMD: GET NL		
2	0.0055...	MAC: ACK(0) [0x18509ff 3->10]		
3	0.0104...	MAC: Singlecast(7) [0x18509ff 3->10] APP: System CMD: REPORT NL (4,5)		
4	0.1294...	MAC: ACK(7) [0x18509ff 10->3]		
5	98.369...	MAC: Singlecast(0) [0x18509ff 10->3] APP: System CMD: DO NL Test (4,5,10)		
6	98.374...	MAC: ACK(0) [0x18509ff 3->10]		
7	98.448...	MAC: Singlecast(13) [0x18509ff 3->4] APP: System CMD: Neighbor NL Test		
8	98.512...	MAC: ACK(14) [0x18509ff 5->3]		
9	98.825...	MAC: Singlecast(15) [0x18509ff 3->10] APP: System CMD: Neighbor NL Test		
10	98.941...	MAC: ACK(15) [0x18509ff 10->3]		
11	98.948...	MAC: Singlecast(8) [0x18509ff 3->10] APP: System CMD: NL Test Done		
12	99.105...	MAC: ACK(8) [0x18509ff 10->3]		
13	123.42...	MAC: Singlecast(0) [0x18509ff 10->3] APP: System CMD: GET NL		
14	123.43...	MAC: ACK(0) [0x18509ff 3->10]		
15	123.43...	MAC: Singlecast(9) [0x18509ff 3->10] APP: System CMD: REPORT NL (4,5,10)		

Figure 79. SDR is adding fake Node 10 to the NL of Node 3.

In frame 1, the NL is requested from Node 3 to demonstrate that Node 10 is currently not a neighbor. The NL is reported at frame 3, having Nodes 4 and 5 as one-hop neighbors. The Do NL Test is sent by the SDR test platform at frame 5, where the targets include the original NL and Node 10. Frames 7-10 are received at R1 and involve Node 3 testing each requested node from lowest to highest node

ID. frame 7 shows Node 3 testing if Node 4 is a one-hop neighbor; however, due to the poor reception rate of the R1 SDR stack, the response is not captured. The test frame sent to Node 5 is not captured either; however, frame 8 shows Node 5 responding to the test request message. Fortunately, a complete test is captured in frames 9 and 10. In frame 9, Node 3 is testing Node 10 as a one-hop neighbor. The SDR test platform responds with an ACK at R1 in frame 10. Node 3 announces that the tests have completed in frame 11. The NL of Node 3 is again requested by Node 10 in frame 13. The response shows that its NL now includes Node 10, along with the original neighbors of Node 4 and 5. Thus, a fake node is added as a neighbor to the NL of a routing node.

This attack may be used to add nodes into the topology without formally conducting network inclusion with the controller. The possibility that a modified NL ends up in the adjacency table depends on the behavior of the controller. The controller used in this network under study sends a `Do NL Test` message before sending a `Get NL` request and only sets the IDs of the nodes that have performed network inclusion in the NL primitive of the NL test message. Thus, rogue insertions get masked out of the target's NL by the controller during initialization and network healing operations.

The attack requires the outsider node to be one hop from the target node. If the outsider is more than one hop away, it is unable to successfully ACK the NL test frame sent over R1. Attempts to route MAC layer ACK frames are silently dropped by the forwarding nodes.

Outsider Topology Discovery.

While the outsider may passively observe routing frames over a long period of time to realize the topology of the target network, a more direct approach is to use the NL request message to learn the NL of every node. Unlike the controller, an

outsider may not be aware of every node in the network. Nodes more than one hop from the outsider require a SR to reach them; however, constructing a SR requires an existing topology model. These problems are addressed using the Topology Discovery algorithm shown in Algorithm 1. In this algorithm, the outsider learns of all of its one-hop neighbors by first broadcasting a **NL Request** message. The outsider collects NL reports of its one-hop neighbors to construct a two-hop topology. In turn, each node in the two-hop topology that has not provided its NL is queried to learn of its neighbors. The outsider uses the two-hop topology model to realize valid SRs to send **NL Request** messages to each target. The responses are added to the two-hop topology model to create a three-hop topology model. Again, the new nodes discovered in the three-hop topology model are queried for their NLs to expand the three-hop model to a four-hop topology model. The algorithm continues until a NL for every discovered node is acquired. Upon termination and assuming the network is not partitioned, the derived topology model represents the target Z-Wave network. Moreover, the outsider realizes its location within the network topology.

Algorithm 1 is implemented in C++ using *libZwave.so* to interact with the network under study. SRs are derived from the topology model using *Dijkstra's Shortest Path* algorithm. Figure 80 provides the console output of the algorithm running on the SDR test platform as Node 10, physically located within the network. The figure is divided into three regions. The top region shows the results of the outsider deriving its NL using a local broadcast request, of which Nodes 1, 4, 5, and 2 provide replies. In the middle section, the algorithm realizes that Node 3 is the only observed node in the two-hop topology model without a known NL. The algorithm sends a request message using a SR derived from the two-hop topology model that routes through Node 4. The NL report from Node 3 indicates it is adjacent to Nodes 4 and 5. After the topology model is updated, the algorithm terminates because every discovered

node has provided its NL.

```
cbadenhop@cbadenhop-Precision-M4600 ~/zwave-routing-dev/worksp
ace/zwave_x $ ./topologyDiscovery
=====Broadcasting NL Req=====
1: 2
4: 2,3
5: 2,3
2: 1,4,5
=====
My NL is: 1,2,4,5
Known nodes: 1 2 3 4 5 10
Known Nls: 1 2 4 5 10
Set Diff: 3
Missing 1 neighbor lists
Querying 3: Generating SR from 10 to 3
Type: 0
Length: 1
Route: [ 4 ]@0
SUCCESS
Missing 0 neighbor lists
=====Topology Model Results=====
1: 2
2: 1,4,5
3: 4,5
4: 2,3
5: 2,3
10: 1,2,4,5
=====
cbadenhop@cbadenhop-Precision-M4600 ~/zwave-routing-dev/worksp
ace/zwave_x $ █
```

Figure 80. Results of the SDR test platform performing the Topology Discovery algorithm on the network under study.

The bottom portion of the figure reports the derived topology model as a list of adjacencies in node ID order. The learned adjacencies between legitimate nodes are consistent with Figure 63, which defines the demonstration topology of the network. Since the reported topology model includes the adjacencies for Node 10, the outsider is able to launch multihop attacks over the network.

Arbitrary SR Cache Modification.

Due to the centralized command and control structure of the routing protocol, nodes are assigned routes using **SR Cache Assignment** messages. The outsider may

use this message to change the routing behavior of target source nodes. The vulnerability is demonstrated on the network under study, and the results are shown in Figure 81. The figure initially shows an application frame taking a route through Node 2 in frames 1-7. In frame 8, the first entry of the source node's routing cache for Node 5 is modified by the outsider. In frames 10-19, another application frame is sent from Node 1 to Node 5. The figure shows the application frame taking the route provided by the outsider.

No.	Time	Info
1	0.0000...	MAC: Singlecast(13) [0x18509fff 1->5] NET: SR [2]@0 APP: Switch Binary
2	0.0130...	MAC: Singlecast(13) [0x18509fff 1->5] NET: SR [2]@1 APP: Switch Binary
3	0.0205...	MAC: Singlecast(13) [0x18509fff 5->1] NET: ACK [2]@0
4	0.0208...	MAC: Singlecast(13) [0x18509fff 1->5] NET: SR [2]@1 APP: Switch Binary
5	0.0208...	MAC: Singlecast(13) [0x18509fff 5->1] NET: ACK [2]@0
6	0.0333...	MAC: Singlecast(13) [0x18509fff 5->1] NET: ACK [2]@f
7	0.0385...	MAC: ACK(13) [0x18509fff 1->2]
8	176.57...	MAC: Singlecast(0) [0x18509fff 10->5] APP: System CMD: SET SR Cache 1.0:[3,4,2]
9	176.58...	MAC: ACK(0) [0x18509fff 5->10]
10	186.33...	MAC: Singlecast(6) [0x18509fff 5->1] NET: SR [3,4,2]@0 APP: Switch Binary
11	186.34...	MAC: Singlecast(6) [0x18509fff 5->1] NET: SR [3,4,2]@1 APP: Switch Binary
12	186.35...	MAC: Singlecast(6) [0x18509fff 5->1] NET: SR [3,4,2]@2 APP: Switch Binary
13	186.36...	MAC: Singlecast(6) [0x18509fff 5->1] NET: SR [3,4,2]@3 APP: Switch Binary
14	186.37...	MAC: Singlecast(6) [0x18509fff 1->5] NET: ACK [3,4,2]@2
15	186.37...	MAC: Singlecast(6) [0x18509fff 1->5] NET: ACK [3,4,2]@2
16	186.38...	MAC: Singlecast(6) [0x18509fff 1->5] NET: ACK [3,4,2]@1
17	186.39...	MAC: Singlecast(6) [0x18509fff 1->5] NET: ACK [3,4,2]@0
18	186.40...	MAC: Singlecast(6) [0x18509fff 1->5] NET: ACK [3,4,2]@f

Figure 81. The routing behavior of Node 5 is modified by the SDR test platform to take an alternate route.

Modification of Routed Frames.

Using the SR cache modification attack, the outsider can direct application frames to flow through a Man-In-The-Middle (MITM) node, where they may be modified. The extent of the manipulation is contingent on the satisfaction of the watchdog node observing the MITM node. Failing to adequately forward a frame results in the watchdog alerting the frame originator with a route NACK. Upon receipt of the NACK, the source node may select an alternative route that does not include the MITM node. Thus, the outsider must avoid invoking any route NACKs to remain stealthy and maximize the lifetime of its advantage [BM14]. For this purpose, several

manipulations are explored to identify conditions where the frame is forwarded by a MITM node without tripping the watchdog.

The case without frame modification is shown in Figure 82. In the figure, a forged routed frame is injected into the network from Node 2 to Node 3, using a route that traverses through Nodes 4 and 10. Acting as the outsider node, the SDR test platform correctly forwards the frame to the destination without alerting the watchdog (i.e., Node 4). The watchdog is satisfied because Node 10 only modifies the hop count field and recalculates the checksum before forwarding. Note that Node 10 is also able to correctly decrement and forward the route ACK in frames 5 and 6 without alerting the watchdog.

No.	Time	Info
1	0.0000...	MAC: Singlecast(0) [0x18509ff 2->3] NET: SR [4,10]@0 APP: Switch Binary
2	0.0071...	MAC: Singlecast(0) [0x18509ff 2->3] NET: SR [4,10]@1 APP: Switch Binary
3	0.0176...	MAC: Singlecast(0) [0x18509ff 2->3] NET: SR [4,10]@2 APP: Switch Binary
4	0.0247...	MAC: ACK(0) [0x18509ff 3->10]
5	0.0305...	MAC: Singlecast(0) [0x18509ff 3->2] NET: ACK [4,10]@1
6	0.0400...	MAC: Singlecast(0) [0x18509ff 3->2] NET: ACK [4,10]@0
7	0.0490...	MAC: Singlecast(0) [0x18509ff 3->2] NET: ACK [4,10]@f
8	0.0605...	MAC: ACK(0) [0x18509ff 2->4]

Figure 82. The SDR test platform is forwarding routed frames without error.

A counter example is provided in Figure 83. Upon receiving frame 2, Node 10 correctly updates the hop index field but fails to recalculate the checksum before forwarding. While the destination receives frame 3, it does not provide an ACK because the received frame does not pass the checksum test. As the watchdog, Node 4 also detects the checksum error and resends the forwarded frame to Node 10 two additional times. Node 10 makes additional transmission attempts in frames 5 and 7. Since the checksum is invalid, these are also ignored by the recipient. After observing three failures, Node 4 generates a route NACK in frame 8, reporting Node 10 (i.e., hop index 1) as the offender.

An outsider node can modify the application payload without triggering the watchdog. With respect to the application layer of a frame, the first two bytes are the most

No.	Time	Info
1	0.0000...	MAC: Singlecast(0) [0x18509ff 2->3] NET: SR [4,10]@0 APP: Switch
2	0.0075...	MAC: Singlecast(0) [0x18509ff 2->3] NET: SR [4,10]@1 APP: Switch
3	0.0195...	[CHKSM ERR]
4	0.0534...	MAC: Singlecast(0) [0x18509ff 2->3] NET: SR [4,10]@1 APP: Switch
5	0.0637...	[CHKSM ERR]
6	0.1404...	MAC: Singlecast(0) [0x18509ff 2->3] NET: SR [4,10]@1 APP: Switch
7	0.1505...	[CHKSM ERR]
8	0.1659...	MAC: Singlecast(0) [0x18509ff 3->2] NET: NACK@1 [4,10]@f
9	0.1787...	MAC: ACK(0) [0x18509ff 2->4]

Figure 83. A route NACK is triggered when the SDR test platform forwards a frame with an invalid checksum.

significant, defining the command class and the command to be executed, respectively. By modifying these bytes, the intent of the frame is drastically changed. To show this on the network under study, the MITM node is modified to write nulls to the first two bytes, turning any frame into a harmless **Hello** message. Again, the test frame is injected into the network and the response is captured in Figure 84. frame 3 shows that Node 10 forwards the injected frame but changes its meaning to be a **Hello** message. Node 3 replies with a route ACK, allowing Node 2 to believe Node 3 received the switch state update. Meanwhile, the watchdog does not detect the tampering of the application layer, so a route NACK is not generated.

No.	Time	Info
1	0.0000...	MAC: Singlecast(0) [0x18509ff 2->3] NET: SR [4,10]@0 APP: Switch Binary
2	0.0078...	MAC: Singlecast(0) [0x18509ff 2->3] NET: SR [4,10]@1 APP: Switch Binary
3	0.0194...	MAC: Singlecast(0) [0x18509ff 2->3] NET: SR [4,10]@2 APP: Hello / NOOP
4	0.0257...	MAC: ACK(0) [0x18509ff 3->10]
5	0.0321...	MAC: Singlecast(0) [0x18509ff 3->2] NET: ACK [4,10]@1
6	0.0424...	MAC: Singlecast(0) [0x18509ff 3->2] NET: ACK [4,10]@0
7	0.0505...	MAC: Singlecast(0) [0x18509ff 3->2] NET: ACK [4,10]@f

Figure 84. MITM Attack: Changing the meaning of a forwarded frame without detection.

Several other fields are modified by the outsider node to test the detection capability of the watchdog, with mixed results. It is discovered that the MITM node may modify the destination or next inner node hop without triggering the watchdog. Conversely, modifying the hop count to have a value beyond the SR length is detected and reported by the watchdog. A rigorous study on the MPDU header, network layer, and application layer fields is required to completely understand the detection rules

of the watchdog.

In addition to the watchdog, the Z-Wave security layer, described in Section 6.5, allows a destination to determine if several fields of a secure frame have been tampered with while being forwarded. Consequently, a MITM modifying the payload or destination field of a secured application frame is detected by the destination upon checking the signature of the frame before the application message is processed. Given the destination finds the frame to be tampered, it does not generate a route ACK. After a timeout is observed by the source node, it retransmits the secure frame. If the MITM attack continues to modify the secure frame, consecutive timeouts allow the source node to conclude the route is flawed. Eventually, the source node selects an alternative SR from its cache. Depending on the topology, the new route may not include the MITM node.

Black Hole Attack.

By exploiting the impersonation, topology discovery, watchdog subversion, and SR cache modification vulnerabilities, a Black Hole attack may be conducted on a Z-Wave network for a given source and destination pair. A Black Hole attack is a frame dropping attack where a node under the influence of the attacker, a Black Hole Node (BHN), silently drops application frames when it is expected to forward them [BRM16]. Active Black Hole attacks exacerbate frame loss by manipulating topology state information in the network to increase the number of routes that flow through the BHN [BM14]. For protocols where routing options are competitive, such as AODV and DSR, the BHN may exaggerate its connectedness to other nodes to provide a more lucrative option than those realized through legitimate route discovery.

The Black Hole attack predicate in [BRMM16] states that a Black Hole attack is possible if the shortest hop distance path between the source node and BHN is

less than the shortest hop distance path between the source and destination node. With respect to Z-Wave, routes are not competitive; rather, they are assigned by the controller. In this case, the predicate is relaxed to where a Black Hole attack is possible if there exists at least one path between the BHN and the source that does not contain the destination node. The path may not contain the destination node because, after appending the destination node to the end of the route, the resulting route is not loop-free. Due to the route length constraint of Z-Wave routing, this path must be no longer than four hops; otherwise, the BHN is unable to append the destination to the end of the route.

To avoid the watchdog of the BHN reporting frames being dropped, the BHN may insert a fictitious hop between the BHN and the destination in the cache assignment message it sends to the source node. In this way, the watchdog is able to observe the BHN forwarding the routed frame as expected without improper modification. At this point, the BHN is supposed to serve as the watchdog for the protocol. Instead of alerting the source to the failure, the BHN allows the frame to be silently lost by the fictitious node. To conduct this variation of the Black Hole attack, a route must exist between the BHN and source node that is no more than three hops and does not contain the destination node. This allows both the BHN and imaginary node to be appended as inner hops of the assigned SR to the destination.

The BHN initiates the attack on the network under study by performing topology discovery to derive a local adjacency table. The table is used to determine if a path exists between the source node and the BHN that is no more than three hops in length and does not contain the destination node. If a path meeting this criteria is found, a **SR Cache Assignment** message is sent to the source node, where the inner hops in the cache entry primitive include all inner hops from the source node to the BHN, the BHN itself, and imaginary Node 11. The target field of the cache assignment message

is set to the destination node ID, the cache index is set to zero, and the status byte is set to 0x10 to give the route the highest preference when the source node selects a route to the destination.

In Figure 85, the attack is demonstrated for a route from Node 5 to Node 1 on the network under study. Prior to the first frame, the BHN (i.e., Node 10) performs the Topology Discovery algorithm. In frame 181, the BHN updates the SR cache of Node 5 using a route that contains both Nodes 10 and 11. Using the human-controller interface, the switch of Node 5 is remotely toggled to invoke network traffic between Nodes 1 and 5. The toggle switch command is routed to Node 5 in frames 183-187. To be sure that the command is honored, Node 1 requests the switch state of Node 5 in frames 188-192. Node 5 generates a reply and routes it through the BHN in frames 193-196. The Black Hole attack is observed at frame 194, where the BHN forwards the frame to non-existent Node 11. To close the loop with the source node, the BHN forges the route ACK from the destination in frame 195 of the figure. Being satisfied that Node 1 received its update, Node 5 makes no attempts to resend its state information using an alternative route.

No.	Time	Proto	Info
179	0.4114...	Zwa...	MAC: Singlecast(11) [0x18509ff 3->10] APP: System CMD: REPORT NL (4,5)
180	0.4165...	Zwa...	MAC: ACK(11) [0x18509ff 10->3]
181	2.4220...	Zwa...	MAC: Singlecast(1) [0x18509ff 10->5] APP: System CMD: SET SR Cache 1.0:[10,11] Valid
182	2.4298...	Zwa...	MAC: ACK(1) [0x18509ff 5->10]
183	24.722...	Zwa...	MAC: Singlecast(13) [0x18509ff 1->5] NET: SR [2]@0 APP: Switch Binary
184	24.734...	Zwa...	MAC: Singlecast(13) [0x18509ff 1->5] NET: SR [2]@1 APP: Switch Binary
185	24.743...	Zwa...	MAC: Singlecast(13) [0x18509ff 5->1] NET: ACK [2]@0
186	24.752...	Zwa...	MAC: Singlecast(13) [0x18509ff 5->1] NET: ACK [2]@f
187	24.757...	Zwa...	MAC: ACK(13) [0x18509ff 1->2]
188	24.837...	Zwa...	MAC: Singlecast(14) [0x18509ff 1->5] NET: SR [2]@0 APP: Switch Binary
189	24.837...	Zwa...	MAC: Singlecast(14) [0x18509ff 1->5] NET: SR [2]@0 APP: Switch Binary
190	24.850...	Zwa...	MAC: Singlecast(14) [0x18509ff 1->5] NET: SR [2]@1 APP: Switch Binary
191	24.856...	Zwa...	MAC: Singlecast(14) [0x18509ff 5->1] NET: ACK [2]@0
192	24.872...	Zwa...	MAC: ACK(14) [0x18509ff 1->2]
193	24.903...	Zwa...	MAC: Singlecast(10) [0x18509ff 5->1] NET: SR [10,11]@0 APP: Switch Binary
194	24.916...	Zwa...	MAC: Singlecast(10) [0x18509ff 5->1] NET: SR [10,11]@1 APP: Switch Binary
195	24.928...	Zwa...	MAC: Singlecast(10) [0x18509ff 1->5] NET: ACK [10,11]@f
196	24.931...	Zwa...	MAC: ACK(10) [0x18509ff 5->10]

Figure 85. Black Hole attack on Z-Wave network where Node 10 is dropping frames sent from Node 5 to Node 1.

The scope of the attack is limited to the network layer. While frames are silently dropped, the application layer expects a response to its query and may repeatedly

attempt to resend the request. Not shown in Figure 85 is that the controller makes several repeated attempts to query the state of Node 5 every 40 seconds. Since no routing errors have been detected, the source node continues to use the route containing the BHN. While the BHN successfully drops the reply to each repeated request, the human controller interface may eventually notify the user of the communication problems with the remote node. This, in turn, may provoke the use of network healing in attempt to remedy the failure.

Recommendations.

Common to all of the discovered vulnerabilities is the exploitation of the hierarchical relationship between the controller and routing nodes. The issue is that routing nodes rely on external direction from a controller or arbitrary neighbor that is trusted without authentication. As a result, the topology and routes can be arbitrarily modified to benefit the attacker. For example, the Black Hole attack may be used on these networks to prevent home security sensors from reporting intrusion events or drop frames that would otherwise actuate an alarm to alert the user. Given these vulnerabilities, several improvements may be made to the protocol.

The primary recommendation is to mandate the use of the *Security* command class for all *System* command class messages. This prevents an outsider who does not have the AES keys from forging messages to control the topology. At a minimum, this policy increases the cost of routing attacks, requiring physical access to a device EEPROM to extract the encryption and authentication keys [BR16].

Given the available global topology information, the controller should be the only node to send SR cache assignments to other devices. Unfortunately, devices are unable to authenticate at the node ID resolution with the current symmetric key capability. A second recommendation is to move to an asymmetric key system, where each node

has a public and private authentication key. The intent for Z-Wave is to be a low cost home automation system [Pae13], so a full Public Key Infrastructure (PKI) may not be feasible. At a minimum, the controller should use an asymmetric key pair so that devices may authenticate messages from the controller. This allows the policy to be enforced, where a device updates its cache only if the source authenticated assignment message originates from the controller. As with the network key [BR16], the controller’s public key may be exchanged with a device during the pairing operation.

In the event that a full PKI implementation exists for Z-Wave, more secure routing is possible. For example, following the practices found in Secure Ad-Hoc On-demand Distance Vector routing (SAODV), the destination may sign the route ACK to make forgery more difficult [M.06]. Moreover, each hop of a source route may, in turn, sign the message to ensure the SR, selected by the source node, is taken. Hop by hop authentication is already found in the Ariadne routing protocol [HPJ02, AY07], which is a more secure form of DSR.

6.6 Conclusion

Being a reverse engineering effort, a complete understanding of the routing protocol, even if achievable, is resource intensive. There are several aspects of the routing protocol still requiring reverse engineering and security analysis. Future work should examine the role of secondary controllers, the existence of a route discovery mechanism similar to DSR as reported in [Pae13], multipath routing, Z-Wave to IP gateway activities, and *System* command class messages not identified in Table 8. Once the proprietary I/O mechanisms of the Z-Wave transceiver chip are identified, static analysis of the firmware may reveal further details of the routing protocol.

As of 2014, new Z-Wave devices may support *Z-Wave Plus*, which is an extended capability beyond the standard Z-Wave protocol. The Z-Wave Alliance has revealed

little about the implications of the capability other than it is to provide “a whole new level of smart home capabilities” using a new hardware platform [Z-W16]. To date, the availability of open source literature and analysis tools, such as an R3 SDR transceiver, is limited for the new technology. None of the devices in the network under study support Z-Wave Plus, so it is not yet known if the vulnerabilities discovered herein extend to the new hardware platform. Until Z-Wave Plus devices become ubiquitous and analysis tools become available, the evaluation of the security implications of Z-Wave Plus is left as future work.

The work performed herein is a result of an observational study on a limited set of Z-Wave devices. While the contributions of this work provide supporting evidence that the Z-Wave protocol has security flaws, conclusive evidence requires a randomized experiment where devices are randomly sampled from the market. Simply evaluating more devices results in a larger observational study with a marginal improvement in scope; therefore, a randomized experiment should be conducted as future work to determine if the discovered vulnerabilities are limited to a particular vendor, firmware version, topology configuration, or hardware version.

The Black Hole attack provided in this work applies to a single source and destination pair. Future work should extend this capability to provide a network-wide attack, which attempts to maximize the number of source and destination pairs affected by the Black Hole attack. With a network-wide attack capability, the applicability of the analytical Black Hole attack model to predict frame loss in Z-Wave networks, developed in [BM14] and [BRMM16], may be explored.

In this chapter, several contributions have been made regarding the security implications of the Z-Wave routing protocol for IoT applications. The forwarding and topology mechanism of the Z-Wave routing protocol are reverse engineered using passive and active observations on a real-world Z-Wave network. A security assessment

of this protocol on the network under study reveals that the hierarchical relationship between the routing nodes and controller may be exploited by a malicious outsider. Not only can an outsider impersonate other nodes, it also has the ability to discover and manipulate the topology, modify route caches of nodes, and manipulate frames in transit. The vulnerabilities are exploited to conduct a Black Hole attack on the network under study, where it is shown that frames are silently discarded for a given source and destination. The Black Hole attack can be used to prevent sensor reports or actuating commands between the controller and devices, inhibiting the functionality of the IoT automation system. The results of the security analysis suggest the Z-Wave routing protocol is vulnerable to integrity-based attacks; however, a more rigorous study is required to characterize the classes of Z-Wave devices possessing the identified vulnerabilities.

Algorithm 1 The **Outsider Topology Discovery** algorithm, where *OutsiderNL*, *KnownNodes*, *HaveNLs*, *MissingNLs* are each a set of node IDs. The topology state is stored in *top*, which is a collection of NLs and is indexed by the owner of the NL. The variable *msg* is a Z-Wave NL **report** frame, where *msg.src* is the source ID and *msg.nl* is the neighbor list included in the frame. The variable *OutsiderID* is a node ID the outsider designates as itself.

```

KnownNodes  $\leftarrow \{OutsiderID\}$ 
HaveNLs  $\leftarrow \{\emptyset\}$ 
OutsiderNL  $\leftarrow \{\emptyset\}$ 
Broadcast_NL_Request()
while !timeout() do
    msg  $\leftarrow$  Receive_NL_Report()
    OutsiderNL  $\leftarrow \{msg.src\} \cup OutsiderNL$ 
    top[msg.src]  $\leftarrow msg.nl$ 
    HaveNLs  $\leftarrow \{msg.src\} \cup HaveNLs$ 
    KnownNodes  $\leftarrow \{msg.src\} \cup KnownNodes$ 
    for all i  $\in msg.nl$  do
        KnownNodes  $\leftarrow \{i\} \cup KnownNodes$ 
    end for
end while
top[OutsiderID]  $\leftarrow OutsiderNL$ 
HaveNLs  $\leftarrow \{OutsiderID\} \cup HaveNLs$ 
NeedNLs  $\leftarrow KnownNodes - HaveNLs$ 
while NeedNLs  $\neq \{\emptyset\}$  do
    for all i  $\in NeedNLs$  do
        Send_NL_Request(top, i)
        msg  $\leftarrow$  Receive_NL_Report(i)
        assert(msg.src = i)
        top[msg.src]  $\leftarrow msg.nl$ 
        HaveNLs  $\leftarrow \{msg.src\} \cup HaveNLs$ 
        for all j  $\in msg.nl$  do
            KnownNodes  $\leftarrow \{j\} \cup KnownNodes$ 
        end for
    end for
    NeedNLs  $\leftarrow KnownNodes - HaveNLs$ 
end while
return top

```

VII. Concluding Remarks

Z-Wave is an IoT implementation, where integration complexity is minimized for its users. This work examined whether or not Z-Wave appropriately provides security and privacy as required in [ITU12b], which has resulted in three journal and two conference publications [BRM16, BRMM16, BGR⁺17, BFH⁺15, BR16].

This work proposes a comprehensive security policy for an IoT device is proposed, where the policy is to obtain and maintain the identified security attributes. As a case study for IoT systems, where Z-Wave is analyzed to identify violations to the security policy. Where necessary, portions of Z-Wave are reverse engineered to facilitate the security evaluation.

A summary of the research findings is shown in Figure 86. In the figure, the chapters providing evidence of a security violation are shown to the right of their corresponding security attributes. Regarding the computational elements, the firmware exploitation attack from Chapter III proves the firmware may be extracted and modified, demonstrating inadequate *anti-tamper* mechanisms. In the same chapter, it is shown that the firmware and non-volatile data memory lack *confidentiality* because they may be extracted. The key extraction attack in Chapter IV demonstrates that even the most critical pieces of data are not properly secured at rest.

The communication components of a Z-Wave IoT device are also examined in this work. Regarding the exchange of Z-Wave frames, Chapter IV shows that, when used, the Z-Wave security layer provides confidentiality, data integrity, and replay protection. However, the key extraction attack demonstrated in Chapter IV shows that the compromise of a single device leads to the loss of this capability for an entire Z-Wave network. Moreover, the security layer only protects vendor-designated portions of the application layer. Chapters II and VI demonstrate the security holes in the communication stack when the security layer is not used. Chapter II shows how

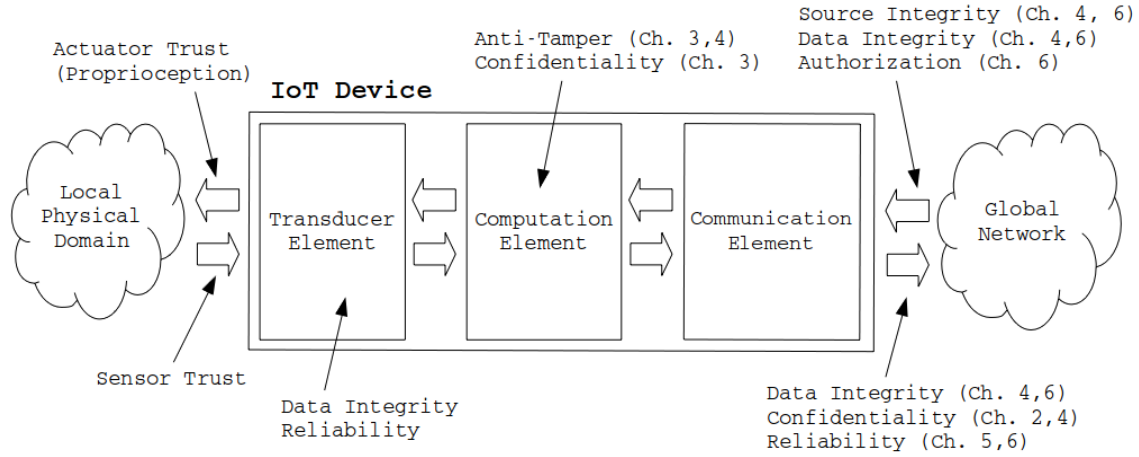


Figure 86. A summary of the results of security analysis on the components of Z-Wave devices with respect to assumed security attributes.

the lack of *confidentiality* in the network inclusion process reveals significant details about device capabilities. In Chapter VI, the network layer is shown to lack *integrity* by demonstrating that the routing mechanism is subject to impersonation and modification. Moreover, an outsider has the capability of controlling non-controller nodes in the network, where only the controller should have this *authority*. Chapter VI uses a Black Hole attack to demonstrate how the corruption of topology state information leads to loss of message delivery *reliability*.

7.1 Future Work

With every research effort, as some questions are answered, more questions are discovered. With respect to the components of a generalized IoT device, Figure 87 reveals which components have been analyzed herein, and which components remain to be studied. The figure indicates that the remaining primary components include the vendor-specific transducer and computational elements. Within the transceiver SoC, the physical layer and the volatile memory of the computational element remain unevaluated. The following subsections propose activities to address these unevaluated

aspects of Z-Wave as future work.

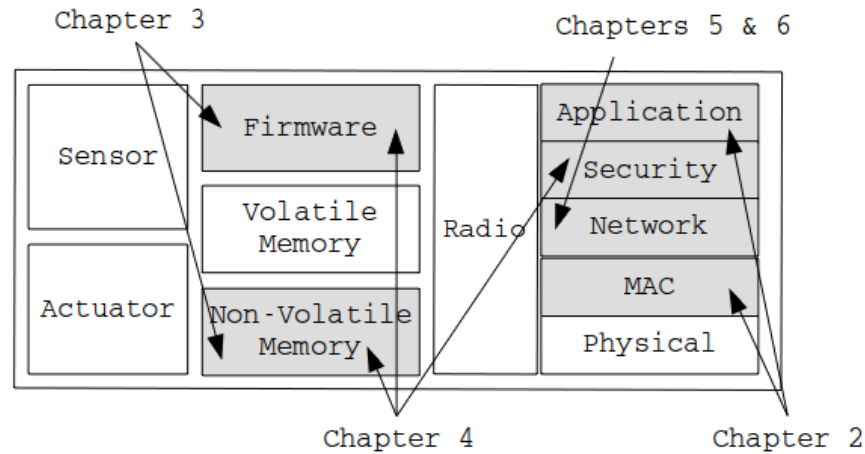


Figure 87. Z-Wave device components covered in this dissertation.

Volatile Memory Analysis.

A major challenge with respect to analyzing volatile memory is its inherent property of being transient. To analyze volatile memory, a dynamic analysis capability, such as a debugger or memory dumper, is required to interrupt normal process flow to peek at memory addresses. Unfortunately, this capability is not yet available to the public for the Z-Wave transceiver SoC. Lacking an apparent hardware debugging capability, dynamic analysis must be achieved through software development of a memory dumping capability. The new capability must be integrated into existing firmware by overcoming three challenges. First, an unused region within the firmware must be identified to hold the new code. Second, the firmware requires modification to redirect program flow to execute the added memory inspection capability. Examples of this include jump patching and function hooking. Third, a proper request and response IO mechanism must be identified. The serial API handler and Z-Wave command class handler are viable candidates, where the latter is preferred. While only controller devices use a serial API handler, all Z-Wave devices use the applica-

tion layer of the radio stack. To utilize a particular IO interface, the associated SFRs must be identified. Currently, the Z-Wave transceiver uses proprietary SFRs for its IO operations, which necessitates additional static analysis reverse engineering of the firmware.

PHY Layer Vulnerabilities.

The Z-Wave physical layer utilizes a narrow-band, so it is likely to be susceptible to narrow-band jamming. The MAC layer drops frames having at least one bit error, which introduces the potential for energy-efficient jamming [BFH⁺15]. In an era of existing low-cost jamming-resistant DSSS transceivers, such as the CC2420 IEEE 802.15.4 radio [TI15], a marginal cost difference may be the only explanation for the design decision. As future work, the effects of narrow-band jamming on Z-Wave transceivers should be explored. As a more compelling case, low-technology RF emitters sharing spectrum with Z-Wave devices could be used to suppress alarm messages of a Z-Wave home security system. A feasibility study should be performed to determine if low-cost Z-Wave devices may be modified to turn one into portable a jammer. One challenge is to suppress the Clear Channel Assessment (CCA) activity predicated the frame emission. Reactive jamming in response to a detected preamble may also be appropriate to be included within this study.

In [ITU12a], the specification states the PHY layer uses a CCA algorithm; however, it does not dictate what algorithm should be used. Through white and black box approaches, the CCA mechanism should be reverse engineered to determine the minimum emitter energy necessary to keep a device in a constant state of waiting for a clear channel.

An energy depletion attack occurs when a target's battery power is rapidly expended by excessive interactions over the radio. In Chapter IV, the batteries of a

door lock are rapidly depleted by repeatedly requesting nonce values. Although this is an energy depletion attack, it is conducted at the application layer, which may leave an audit trail of the attack in activity log files. In future work, PHY layer attacks towards energy depletion should be explored. For example, it may be possible to keep a receiver active indefinitely by sending a preamble sequence that never terminates with a SOF. Incidentally, this may also prevent transmission, achieving a jamming capability.

Trust in Evolving and Heterogeneous Z-Wave Device Configurations.

With respect to the vendor-specific computational and transducer elements, one interesting area of research is to explore the security implications of coordinating processing elements with heterogeneous interconnects. Often trust is implicit between processing elements of an embedded system having interconnected interfaces such as UART, SPI, General Purpose I/O (GPIO) and Inter-Integrated Circuit (I2C). Attestation mechanisms to evaluate the integrity of neighboring components may permit adaptive techniques to minimize privilege escalation via island hopping attack. Conversely, Z-Wave devices may be evaluated on their susceptibility to island hopping attacks.

Z-Wave is an evolving technology. As time progresses, new transceiver SoC hardware, firmware, and communication protocol may become available. New device types, facilitating roles currently unavailable to home and office automation, may be placed in the market. While the new devices introduce additional capabilities, they may also be required to support the legacy aspects of Z-Wave to maintain low integration overhead. As Z-Wave technology changes, future work should examine the implications of these changes to determine if they improve or reduce the security state.

7.2 Results of the Case Study

The Z-Wave case-study shows several challenges to IoT systems. While proprietary restrictions offer some security through obscurity, this *security* is lost once a reverse engineering effort is performed. Although reverse engineering is resource intensive, it is not at the level of brute-force guessing encryption keys. Moreover, once the obscured technical data is made public, it cannot be again obscured without a redesign. Alternatively, if an encryption key is compromised, security may be recovered by simply distributing a new key. Therefore, Z-Wave designers should turn to methods of code signing and encrypting non-volatile data in lieu of security by obscurity. Moreover, security should be layered, whereas Z-Wave security is thinly spread, covering some of the components.

Additionally, this study exposes an issue of distributed system trust, which is presented as an issue in [ACH15]. In Z-Wave networks, end-node devices implicitly trust controller commands, which are subject to impersonation and manipulation. For example, most of the attacks revealed in Chapter VI use the same mechanisms available to the controller to manage the network. This allows a network to be controlled by an outsider, which may even isolate the controller from the rest of the network. End-node devices should not implicitly trust external commands. Each device should be capable of managing its own topology state. Arriving commands should be met with skepticism. Not only should the device be able to go verify the identity of the source node and verify authorization, the end-node device should also be capable of examining its own state to determine if the command *should* be followed. In [ITU12b], one of the requirements is an autonomic capability. Holistically, Z-Wave provides an autonomic capability at the system level; however, it lacks autonomic behavior at the device level.

A final issue regarding IoT security is system complexity, which negatively im-

pacts security design and implementation [RMCO16]. From a design perspective, it is difficult to prove various security properties of a heterogeneous, evolving, and distributed system. To the user, an IoT system is an abstract service deployed in a home or office. In the case of Z-Wave, the user may make decisions that are detrimental to the overall security state of the system. For example, Z-Wave allows home security systems to share the same network as mood lighting. While the vendors of the home security devices may take extra measures to ensure their devices are secure, the mood lighting vendors may not. An attacker may exploit the insecurity of the mood lighting devices to gain access to the network that is also utilized by the home security system. Federating IoT systems is one of the suggested solutions to IoT security vulnerabilities in [MYAZ15]. The distributed system may be partitioned based on required security properties, application function, or safety criticality.

The issue of heterogeneity is prevalent in other embedded system industries, such as the automotive industry. Heterogeneous automotive subsystems are interconnected through a shared Controller Area Network (CAN) bus. Currently, safety critical subsystems share the same bus as the car media center. In [CMK⁺11], the media center of a car is exploited to gain access to more critical subsystems. A *functional separation* approach is taken in [CG17] to protect the safety critical subsystems from non-critical components, which may lack security.

Z-Wave and IoT have significant challenges to overcome before they may safely, reliably, and securely provide home and office services. While the Z-Wave Alliance has made its system appeal to a large consumer base by reducing the integration effort, these DIY installers may be unaware that they are exposing their home and office locations to cyber attack. The attack vectors identified for Z-Wave systems have a wide variety of effects including privacy violations [BFH⁺15], perimeter intrusion [BR16], property damage [HR16], alarm suppression, and general mischief. While

this effort has exposed many of the vulnerabilities and attack vectors, the exposure through publication is limited to the academic and hacker environments. It is the responsibility of the Z-Wave device and transceiver SoC developers to properly address these vulnerabilities with updates to their proprietary firmware and hardware. They should also properly educate DIY installers of the security risks in advertising, packaging, and installation documentation. Moreover, the Z-Wave Alliance should establish and maintain a public security policy. Device vendors using the Z-Wave logo must be certified in adhering to the official security policy. To facilitate different device types and services, several security policies may be necessary. Finally, the consumer must be able to ascertain a given device's compliance through a trusted authority before device acquisition and during its life-cycle.

Bibliography

- ACH15. I. Andrea, C. Chrysostomou, and G. Hadjichristofi. Internet of things: Security vulnerabilities and challenges. In *2015 IEEE Symposium on Computers and Communication (ISCC)*, pages 180–187, July 2015.
- Aeo17. Aeon labs smart energy switch - smart things support. <https://support.smartthings.com/hc/en-us/articles/202294170-Aeon-Labs-Smart-Energy-Switch>, 2017. Last visited May, 2017.
- AFI15. AFITWiSec. Afitwisec github. <https://github.com/AFITWiSec>, 2015. [accessed 2.10.16].
- AH13. L. Apa and C. Hollman. Compromising industrial facilities from 40 miles away. Presented at Blackhat USA, August 2013.
- AHK08. I. Aad, J.-P. Hubaux, and E. W. Knightly. Impact of denial of service attacks on ad hoc networks. *IEEE/ACM Transactions on Networking*, 16(4):791–802, 2008.
- ANS13. A. K. Abdelaziz, M. Nafaa, and G. Salim. Survey of routing attacks and countermeasures in mobile ad hoc networks. In *The 15th International Conference on Computer Modelling and Simulation (UKSim)*, pages 693–698, 2013.
- AS11. A. S. Al Shahrani. Rushing attack in mobile ad hoc networks. In *The Third International Conference on Intelligent Networking and Collaborative Systems (INCoS)*, pages 752–758, 2011.
- AY06. T. R. Andel and A. Yasinsac. On the credibility of manet simulations. *Computer*, 39(7):48–54, 2006.
- AY07. T. Andel and A. Yasinsac. Surveying security analysis techniques in MANET routing protocols. *IEEE Communications Surveys and Tutorials*, vol. 9(no. 4):pp. 70–84, 2007.
- BBS09. A. Bala, M. Bansal, and J. Singh. Performance analysis of manet under blackhole attack. In *The First International Conference on Networks and Communications*, pages 141–145, 2009.
- BBTR15. T. Bihl, K. Bauer, M. Temple, and B. Ramsey. Dimensional reduction analysis for physical layer device fingerprints with application to Zigbee and Z-Wave devices. In *MILCOM*, pages 360–365, 2015.

- BE03. C. Bettstetter and J. Eberspacher. Hop distances in homogeneous ad hoc networks. In *The 57th IEEE Semiannual Vehicular Technology Conference*, volume 4, pages 2286–2290 vol.4, 2003.
- BFH⁺15. C. Badenhop, J. Fuller, J. Hall, B. Ramsey, and M. Rice. Evaluating ITU-T G.9959: Wireless systems in the critical infrastructure. In J. Butts and S. Shenoi, editors, *Critical Infrastructure Protection IX, IFIPS WG 11.10*, pages 61–79. Springer, 2015.
- BGR⁺17. C. Badenhop, S. Graham, B. Ramsey, B. Mullins, and L. Mailloux. The z-wave routing protocol and its security implications. *Elsevier Journal of Computers & Security*, vol. 68:pp. 112–129, July 2017.
- BKPK16. S. G. Bae, Y. Kim, Y. Park, and C. Kim. 3-gb/s high-speed true random number generator using common-mode operating comparator and sampling uncertainty of d flip-flop. *IEEE Journal of Solid-State Circuits*, PP(99):1–6, 2016.
- BL73. D. Bell and L. LaPadula. Secure computer systems: Mathematical foundations. Technical Report MTR-2547, MITRE Corporation, 1973.
- BM14. C. W. Badenhop and B. E. Mullins. A black hole attack model using topology approximation for reactive ad-hoc routing protocols. *International Journal of Security and Networks*, 9(2):63–77, 2014.
- BN89. D. F. C. Brewer and M. J. Nash. The chinese wall security policy. In *IEEE Symposium on Security and Privacy*, pages 206–214, 1989.
- BR16. 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.
- BRM16. C. W. Badenhop, B. W. Ramsey, and B. E. Mullins. An analytical black hole attack model using a stochastic topology approximation technique. *International Journal of Network Security*, 18(4):667–677, 2016.
- BRMM16. 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. *Digital Investigation*, 17:14 – 27, 2016.
- BRR15. A. Boztas, A.R.J. Riethoven, and M. Roeloffs. Smart TV forensics: Digital traces on televisions. *Digital Investigation, Elsevier*, 12(S1):S72–S80, March 2015.
- BS09. N. Bhalaji and A. Shanmugam. Association between nodes to combat blackhole attack in dsr based manet. In *The International Conference on Wireless and Optical Communications Networks*, pages 1–5, 2009.

- CCS13. A. Cui, M. Costello, and S. Stolfo. When firmware modifications attack: A case study of embedded exploitation. In *NDSS Symposium*, April 2013.
- CG17. E. Caberto and S. Graham. A method of securing a vehicles controller area network. In *Cyber Warfare and Security (ICCWS 2017), 12th International Conference on*, March 2017.
- CJ14. K. Chadha and S. Jain. Impact of black hole and gray hole attack in aodv protocol. In *Recent Advances and Innovations in Engineering (ICRAIE)*, pages 1–7, 2014.
- Cla88. D. Clark. The design philosophy of the darpa internet protocols. *SIGCOMM*, 18(4):106–114, 1988.
- CMCJ04. B. Chevallier-Mames, M. Ciet, and M. Joye. Low-cost solutions for preventing simple side-channel analysis: side-channel atomicity. *Computers, IEEE Transactions on*, 53(6):760–768, June 2004.
- CMK⁺11. S. Checkoway, D. Mccoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, and T. Kohno. Comprehensive experimental analyses of automotive attack surfaces. In *USENIX SECURITY*. USENIX, 2011.
- CR16. K. Cooper and B. Ramsey. Exploiting a smart fridge: A case study in kinetic cyber. In *Presented at DEFCON 24*, 2016.
- Cre15. A. Crenshaw. Tsa master key duplication & why ”security through (not so) obscurity” fails - trustedsec. <https://www.trustedsec.com/august-2015/master-key-duplication/>, August 2015. Last accessed March 2017.
- CTK16. A. Chaudhary, V. N. Tiwari, and A. Kumar. A new intrusion detection system based on soft computing techniques using neuro-fuzzy classifier for packet dropping attack in manets. *International Journal of Network Security*, 18(3):514–522, 2016.
- Des16. Sigma Designs. *Security 2 Command Class, version 0.9*, August 2016. [accessed 1.25.17].
- DRXS15. M. Davies, H. Read, C. Xynos, and I. Sutherland. Forensic analysis of a Sony PlayStation 4: A first look. *Digital Investigation, Elsevier*, 12(S1):S81–S89, March 2015.
- EKA15. S. Elbouanani, M. Kiram, and O. Achbarou. Introduction to the internet of things security: Standardization and research challenges. In *2015 11th International Conference on Information Assurance and Security (IAS)*, pages 32–37, December 2015.

- Eve17. An145 wireless screw-in on/off module. <http://www.everspring.com/AN145.aspx>, 2017. Last visited May, 2017.
- FG13. B Fouladi and S. Ghanoun. Security evaluation of the Z-Wave wireless protocol. Presented at Blackhat USA, July 2013.
- For15. FortrezZ. *WV-01 Wireless Z-Wave Water Valve Owner's Manual and Installation Guide*. FortrezZ, 2015.
- Fou17. Wireshark Foundation. Wireshark - go deep. <https://www.wireshark.org/>, May 2017. [accessed 05.11.17].
- FR15. J. Fuller and B. Ramsey. Rogue Z-Wave controllers: A persistent attack channel. In *SenseApp*. IEEE, 2015.
- Fre08. Freescale beestack application development guide rev 1.1. http://cache.freescale.com/files/rf/_if/doc/user/_guide/BSADG.pdf, 2008. Last accessed January 2015.
- FRRP17. J. Fuller, B. Ramsey, M. Rice, and J. Pecarina. Misuse-based detection of z-wave network attacks. *Elsevier Journal of Computers & Security*, vol. 64:pp. 44–58, 2017.
- FS03. N. Ferguson and B. Schneier. *Practical Cryptography*. Wiley, Indianapolis, IN USA, 2003.
- FW10. X. Fei and W. Wenye. On the survivability of wireless ad hoc networks with node misbehaviors and failures. *IEEE Transactions on Dependable and Secure Computing*, 7(3):284–299, 2010.
- GBHF01. D. M. Gregg, W. J. Blackert, D. V. Heinbuch, and D. Furnanage. Assessing and quantifying denial of service attacks. In *Military Communications Conference*, volume 1, pages 76–80 vol.1, 2001.
- GES17. Ge z-wave plug-in outdoor smart switch. <https://byjasco.com/products/ge-z-wave-plug-outdoor-smart-switch>, 2017. Last visited May, 2017.
- Goo15. T. Goodspeed. GoodFET homepage. <http://www.goodfet.sourceforge.net>, December 2015. [accessed 12.15.15].
- GS16. K. Gupta and S. Shukla. Internet of things: Security challenges for next generation networks. In *2016 International Conference on Innovation and Challenges in Cyber Security (ICICCS-INBUSH)*, pages 315–318, February 2016.

- Hed09. O.-E. Hedenstad. Security model for resource availability - subject and object type enforcement. In *Military Communications Conference*, pages 1–7, 2009.
- HHF⁺04. Y. Hao, L. Haiyun, Y. Fan, L. Songwu, and Z. Lixia. Security challenges in mobile ad hoc networks: Challenges and solutions. *IEEE Wireless Communications*, February 2004.
- HHS⁺06. Y. Hong, Z. Hui, K. Sripanidkulchai, S. Zon-Yin, and S. Debanjan. Information leak vulnerabilities in sip implementations. *IEEE Network*, 20(5):6–13, September 2006.
- Hig17. D. Hightower. Sigma Designs acquires Zensys holdings. <http://mwrfl.com/content/sigma-designs-acquires-zensys-holdings>, March 2017. [accessed 3.2.17].
- HMT06. K. J. Hole, V. Moen, and T. Tjostheim. Case study: online banking security. *IEEE Security Privacy*, 4(2):14–20, March 2006.
- HPJ02. Y. C. Hu, A. Perrig, and D. B. Johnson. Ariadne: A secure on-demand routing protocol for ad hoc networks. In *The 8th Annual International Conference on Mobile Computing and Networking*, pages 12–23, 2002.
- HQ12. H. Hejiao and Z. Qiang. Petri-net-based modeling and resolving of black hole attack in wmn. In *The 36th Annual Computer Software and Applications Conference Workshops (COMPSACW)*, pages 409–414, 2012.
- HR16. J. Hall and B. Ramsey. Breaking bulbs briskly by bogus broadcasts. In *Presented at ShmooCon, Washington DC*, 2016.
- HRRL16. J. Hall, B. Ramsey, M. Rice, and T. Lacey. Z-wave network reconnaissance and transceiver fingerprinting using software-defined radios. In *International Conference on Cyber Warfare and Security*, pages 163–171, 2016.
- HXSGLA10. X. Hui, W. Xianren, H. R. Sadjadpour, and J. J. Garcia-Luna-Aceves. A unified analysis of routing protocols in manets. *IEEE Transactions on Communications*, 58(3):911–922, 2010.
- Ins17. Internet enabled raspberry pi pet feeder. <http://instructables.com/id/Internet-Enabled-Raspberry-Pi-Pet-Feeder/>, 2017. Last visited March, 2017.
- Int82. Intel. External product specification for the MCS-51 object module format. Technical report, Intel, September 1982.

- ITU12a. ITU. ITU recommendation G.9959: Short range narrow-band digital radiocommunication transceivers - PHY and MAC layer specifications. Technical report, Telecommunication Standardization Sector of ITU, 2012.
- ITU12b. ITU. ITU recommendation Y.2060: Overview of the Internet of things. Technical report, Telecommunication Standardization Sector of ITU, 2012.
- Jar15. A. Jarno. SDCC - Small Device C Compiler. <http://sdcc.sourceforge.net/>, October 2015. [accessed 10.28.15].
- Jas17. GE z-wave wireless lighting control lamp module with dimmer control. <https://byjasco.com/products/ge-z-wave-wireless-lighting-control-lamp-module-dimmer-control>, 2017. Last visited May, 2017.
- JDB12. R. K. Jha, U. D. Dalal, and I. Z. Bholebawa. Performance analysis of black hole attack on wimax-wlan interface network. In *The Third International Conference on Computer and Communication Technology (ICCCCT)*, pages 303–308, 2012.
- JG08. J. Johansson and R. Grimes. Is security by obscurity a valid approach? <https://technet.microsoft.com/en-us/library/2008.06.obscurity.aspx>, 2008. Last accessed March 2017.
- JHM07. D. Johnson, Y. Hu, and D. Maltz. The dynamic source routing protocol (dsr) for mobile ad hoc networks. Technical Report RFC-4728, IETF, 2007.
- JPY⁺09. C. Jiwen, Y. Ping, T. Ye, Z. Yongkai, and L. Ning. The simulation and comparison of routing attacks on dsr protocol. In *The 5th International Conference on Wireless Communications, Networking and Mobile Computing*, pages 1–4, 2009.
- Keil15a. Keil. Cx51 user’s guide: Variable initialization code. http://www.keil.com/support/man/docs/c51/c51_ap_init.htm, November 2015. [accessed 11.18.15].
- Keil15b. Keil. Keil C51 C compiler. <http://www.keil.com/c51/c51.asp>, October 2015. [accessed 10.28.15].
- KHL⁺16. M. Kim, U. Ha, Y. Lee, K. Lee, and H. J. Yoo. A 82nw chaotic-map true random number generator based on sub-ranging sar adc. In *ESSCIRC Conference 2016: 42nd European Solid-State Circuits Conference*, pages 157–160, 2016.

- Kil16. Github - joswr1ght/killerzee: Killerzee: Tools for attacking and evaluating z-wave networks. <https://github.com/joswr1ght/killerzee>, 2016. Last visited September, 2016.
- KL07. D. Kiwior and L. Lam. Routing protocol performance over intermittent links. In *Military Communications Conference*, pages 1–8, 2007.
- KSSH09. A. Konig, D. Seither, R. Steinmetz, and M. Hollick. An analytical model of routing, misbehavior, and countermeasures in mobile ad hoc networks. In *Global Telecommunications Conference*, pages 1–6, 2009.
- KYSL14. S. Kwon, H. Yoo, T. Shon, and G. Lee. Scenario-based attack route on industrial control system. In *2014 International Conference on IT Convergence and Security (ICITCS)*, pages 1–3, October 2014.
- KZA⁺16. W. Khan, H. Zangoti, M. Aalsalem, M. Zahid, and Q. Arshad. Mobile rfid in internet of things: Security attacks, privacy risks, and countermeasures. In *2016 International Conference on Radar, Antenna, Microwave, Electronics, and Telecommunications (ICRAMET)*, pages 36–41, October 2016.
- Lar14. J. Larson. Miniaturization. Presented at Blackhat USA, August 2014.
- LH12. X. Liying and L. Huiyun. Side channel analysis attacks against channel state information based key generation. In *Systems and Informatics (ICSAI), 2012 International Conference on*, pages 1529–1532, May 2012.
- Lip82. S. Lipner. Non-discretionary controls for commercial applications. In *Symposium on Privacy and Security*, pages 2–10, 1982.
- LMM⁺12. T. Lacey, R. Mills, B. Mullins, R. Raines, M. Oxley, and S. Rogers. Ripsec - using reputation-based multilayer security to protect manets. *Computers & Security, Elsevier*, pages 122–136, 2012.
- M.06. Guerrero-Zapata M. Secure on-demand distance vector (saodv) routing. Technical Report Expired Draft, IETF, 2006.
- MBC16. A. Mayzaud, R. Badonnel, and I. Chrisment. A taxonomy of attacks in rpl-based internet of things. *International Journal of Network Security*, 18(3):459–473, 2016.
- McK12. M. McKay. Best practices in automation security. In *2012 IEEE-IAS/PCA 54th Cement Industry Technical Conference*, pages 1–15, May 2012.
- Met96. MetaLink. *8051 Cross Assembler User's Manual*. MetaLink Corporation, January 1996.

- Mic05. Microchip. *25Aa256/25LC256 256 K SPI Bus Serial EEPROM*. Microchip, 2005.
- MLM17. A. Mesbah, J. L. Lanet, and M. Mezghiche. Reverse engineering a java card memory management algorithm. *Elsevier Journal of Computers & Security*, vol. 66:pp. 97–114, 2017.
- Mot03. Motorola. *SPI Block Guide*. Motorola, 3.06 edition, February 2003.
- MPAM15. C. Matthies, L. Pirl, A. Azodi, and C. Meinel. Beat your mom at solitaire #x2014; a review of reverse engineering techniques and counter-measures. In *2015 6th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, pages 1094–1097, Sept 2015.
- MYAZ15. R. Mahmoud, T. Yousuf, F. Aloul, and I. Zualkernan. Internet of things (iot) security: Current status, challenges and prospective measures. In *2015 10th International Conference for Internet Technology and Secured Transactions (ICITST)*, pages 336–341, December 2015.
- NGFB15. R. Neisse, Steri G., I.N. Fovino, and G. Baldini. Seckit: A model-based security toolkit for the internet of things. *Computers & Security, Elsevier*, pages 1–17, June 2015.
- NKSK02. Swetha Narayanaswamy, Vikas Kawadia, R. S. Sreenivas, and P. R. Kumar. Power control in ad-hoc networks: Theory, architecture, algorithm and implementation of the compow protocol. In *in European Wireless Conference*, pages 156–162, 2002.
- Ope15a. OpenZWave. Openzwave Defs.h. http://www.openzwave.com/dev/Defs_8h_source.html, November 2015. [accessed 11.18.15].
- Ope15b. OpenZWave. Openzwave homepage. <http://www.openzwave.com/home>, October 2015. [accessed 10.28.15].
- Oxy17. Mit project oxygen. <http://oxygen.csail.mit.edu/Overview.html>, 2017. Last visited March, 2017.
- PA12. K. Pavani and D. Avula. Performance evaluation of mobile adhoc network under black hole attack. In *The International Conference on Software Engineering and Mobile Application Modelling and Development (ICSEMA)*, pages 1–6, 2012.
- Pae13. C. Paetz. *Z-Wave Basics*. CreateSpace Independent Publishing Platform, Lexington, KY, 2013.
- Pal13. A. Palmer. Security through obscurity - fail. <https://www.adampalmer.me/iodigitalsec/2013/07/03/>

- security-through-obscurity-fail/, July 2013. Last accessed March 2017.
- PBRD03. C. Perkins, E. Belding-Royer, and S. Das. Ad hoc on-demand distance vector (aodv) routing. Technical Report RFC-3561, IETF, 2003.
- Pic14. J.M. Picod. Dumping firmware out of a z-wave asic. <http://blog.cassidiancybersecurity.com/post/2014/02/Dumping-firmware-from-ASIC>, February 2014. [accessed 2.10.16].
- PLD14. J. Picod, A. Lebrun, and J. Demay. Bringing software defined radio to the penetration testing community. Presented at Blackhat USA, August 2014.
- PR15. J. Patel and B. Ramsey. Comparison of parametric and non-parametric statistical features for Z-Wave fingerprinting. In *MILCOM*, pages 378–382, 2015.
- PSM11. N. Purohit, R. Sinha, and K. Maurya. Simulation study of black hole and jellyfish attack on manet using ns3. In *The Nirma University International Conference on Engineering (NUiCONE)*, pages 1–5, 2011.
- QZJX09. F. Qian, C. Zhongmin, Y. Jin, and H. Xunchao. A performance comparison of the ad hoc network protocols. In *The Second International Workshop on Computer Science and Engineering*, volume 2, pages 293–297, 2009.
- Res99. E. Rescorla. Diffie-hellman key agreement method. Technical Report RFC-2631, IETF, 1999.
- RMCO16. R. Ross, M. McEvilly, and J. Carrier Owen. *Systems Security Engineering: Considerations for a Multidisciplinary Approach in the Engineering of Trustworthy Secure Systems*. NIST, 800-160 edition, November 2016.
- Roh15. S. Rohan. Home automation and control market worth \$12.81 billion by 2020. <http://www.marketsandmarkets.com/PressReleases/home-automation-control-systems.asp>, November 2015. [accessed 11.18.15].
- RSM⁺15. B. Ramsey, T. Stubbs, B. Mullins, M. Temple, and M. Buckner. Wireless infrastructure protection using low-cost radio frequency fingerprinting receivers. *International Journal of Critical Infrastructure Protection*, 8:27–39, 2015.
- rw15. rfwireless world. z-wave protocol stack — z-wave protocol layer basics. <http://www.rfwireless-world.com/Tutorials/z-wave-protocol-stack.html>, December 2015. [accessed 12.15.15].

- SC12. R. K. Sahu and N. S. Chaudhari. Performance evaluation of ad hoc network under black hole attack. In *World Congress on Information and Communication Technologies (WICT)*, pages 780–784, 2012.
- Sca16. Github - bastilleresearch/scapy-radio: scapy-radio (from original hg repo). <https://github.com/BastilleResearch/scapy-radio>, 2016. Last visited September, 2016.
- Sch14. B. Schneier. The insecurity of secret it systems. https://www.schneier.com/blog/archives/2014/02/the_insecurity_2.html, February 2014. Last accessed March 2017.
- SDM12. L. Schauer, F. Dorfmeister, and M. Maier. Potentials and limitations of wifi-positioning using time-of-flight. In *Indoor Positioning and Indoor Navigation (IPIN), 2013 International Conference on*, pages 28–31, October 2012.
- SGA07. M. Stutton, A. Greene, and P. Amini. *Fuzzing: Brute Force Vulnerability Discovery*. Addison-Wesley, Upper Saddle River, NJ USA, 2007.
- Sig15. SigmaDesigns. Dev kits - Z-Wave - Sigma Designs. http://z-wave.sigmadesigns.com/dev_kits#z-wave_whats_in_kit, November 2015. [accessed 11.18.15].
- SKB14. K. Saghar, D. Kendall, and A. Bouridane. Application of formal modeling to detect black hole attacks in wireless sensor network routing protocols. In *The 11th International Bhurban Conference on Applied Sciences and Technology (IBCAST)*, pages 191–194, 2014.
- SL02. E. Skoudis and T. Liston. *Counter Hack Reloaded: A Step-by-Step Guide to Computer Attacks and Effective Defenses*. Prentice Hall, Upper Saddle River, NJ USA, 2nd edition, 2002.
- SPWS13. L. Szekeres, M. Payer, T. Wei, and D. Song. Sok: Eternal war in memory. In *2013 IEEE Symposium on Security and Privacy*, pages 48–62, May 2013.
- SS88. Y. Saad and M. H. Schultz. Topological properties of hypercubes. *IEEE Transactions on Computers*, pages 867–872, 1988.
- SS07. H. Suoto and S. B. Sen. Cluster-based region formation for modeling manet. In *Military Communications Conference*, pages 1–7, 2007.
- SSD12. M. Salehi, H. Samavati, and M. Dehghan. Evaluation of dsr protocol under a new black hole attack. In *The 20th Iranian Conference on Electrical Engineering (ICEE)*, pages 640–644, 2012.

- SSD14. K. J. Sarma, R. Sharma, and R. Das. A survey of black hole attack detection in manet. In *The International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT)*, pages 202–205, 2014.
- Ste05. C. Steiner. *The 8051/8052 Microcontroller*. Universal Publishers, Boca Raton, FL USA, 2005.
- Ste15. B. Stewart. *Adventures in Arduino*. Wiley, West Sussex, UK, 2015.
- STM16. Sarbpreet, S. Tripathy, and J. Mathew. Design and evaluation of an iot enabled secure multi-service ambulance tracking system. In *2016 IEEE Region 10 Conference (TENCON)*, pages 2209–2214, November 2016.
- SVM15. J. Stuttgen, S. Vomel, and Denzel M. Acquisition and analysis of compromised firmware using memory forensics. *Digital Investigation, Elsevier*, 12(S1):S50–S60, March 2015.
- SYJ10. Z. Siyu, P. Yongxiang, Y. Yang, and L. Jianping. An open architecture for the routing protocols design in ad hoc networks. In *The 3rd IEEE International Conference on Computer Science and Information Technology (ICCSIT)*, volume 7, pages 18–22, 2010.
- TACC09. M. Tiwari, V. Arya, R. Choudhari, and K. Choudhary. Designing intrusion detection to detect black hole and selective forwarding attack in WSN based on local information. In *International Conference on Computer Sciences and Convergence Information Technology*, pages 824–828, 2009.
- TETH16. M. Tellez, S. El-Tawab, and M. H. Heydari. Iot security attacks using reverse engineering methods on wsn applications. In *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, pages 182–187, Dec 2016.
- Tho53. R. L. Thorndike. Who belongs in the family? *Psychometrika*, 18(4):267–276, 1953.
- TI14. Low-power sub-1 ghz rf transceiver. Technical report, Texas Instruments, 2014.
- TI15. 2.4 ghz ieee 802.15.4 / zigbee-ready rf transceiver. Technical report, Texas Instruments, 2015.
- Ver17. Vera: Smarter home control. <http://getvera.com>, 2017. Last visited May, 2017.

- ves15. vesternet. Understanding z-wave networks, nodes & devices. <http://www.vesternet.com/resources/technology-indepth/understanding-z-wave-networks>, December 2015. [accessed 12.15.15].
- YB11. J. Yu and P. Brune. No security by obscurity - why two factor authentication should be based on an open design. In *Proceedings of the International Conference on Security and Cryptography*, pages 418–421, July 2011.
- YM03. P. Yau and C. Mitchell. Security vulnerabilities in ad hoc networks. In *The 7th International Symposium on Communication Theory and Applications*, pages 148–153, 2003.
- YYH⁺11. F. Yanfei, J. Yixin, Z. Haujin, C. Jiming, and S. Xuemin. Network coding based privacy preservation against traffic analysis in multi-hop wireless networks. *Wireless Communications, IEEE Transactions on*, 10(3):834–843, March 2011.
- YZV14. Z. Yan, P. Zhang, and A. Vasilakos. A survey on trust management for internet of things. *Network and Computer Applications, Elsevier*, 42:120–134, 2014.
- Z-S16. Z-Wave USB stick. <http://aeotec.com/z-wave-usb-stick>, 2016. Last visited September, 2016.
- Z-W16. Z-wave plus. <http://www.zwaveproducts.com/learn/z-wave/z-wave-plus>, 2016. Last visited September, 2016.
- Z-W17. Z-wave the smartest choice for your smart home. <http://www.z-wave.com>, 2017. Last visited March, 2017.
- Zen07. Zensys. *ZM3102N Z-Wave Module Datasheet*. Zensys, October 2007.
- ZG12. S. A. R. Zaidi and M. Ghogho. Stochastic geometric analysis of black hole attack on smart grid communication networks. In *The Third International Conference on Smart Grid Communications (SmartGridComm)*, pages 716–721, 2012.
- Zwa16. z-wave protocol stack — z-wave protocol layer basics. <http://www.rfwireless-world.com/Tutorials/z-wave-protocol-stack.html>, 2016. Last visited September, 2016.
- ZWA17. The internet of things is powered by z-wave. <http://www.z-wavealliance.org>, 2017. Last visited March, 2017.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. **PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.**

1. REPORT DATE (DD-MM-YYYY) 15-06-2017			2. REPORT TYPE Dissertation		3. DATES COVERED (From — To) October 2014 — June 2017	
4. TITLE AND SUBTITLE A Multifaceted Security Evaluation of Z-Wave, a Proprietary Implementation of the Internet of Things					5a. CONTRACT NUMBER	
					5b. GRANT NUMBER	
					5c. PROGRAM ELEMENT NUMBER	
6. AUTHOR(S) Badenhop, Christopher W.					5d. PROJECT NUMBER JON: 16-187	
					5e. TASK NUMBER	
					5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 Hobson Way WPAFB OH 45433-7765					8. PERFORMING ORGANIZATION REPORT NUMBER AFIT-ENG-DS-17-J-074	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Laboratory 2241 Avionics Circle WPAFB OH 45433 DSN 713-8140, COMM 937-713-8140 Email: juan.carbonell@us.af.mil					10. SPONSOR/MONITOR'S ACRONYM(S) AFRL/RDWD	
					11. SPONSOR/MONITOR'S REPORT NUMBER(S)	
12. DISTRIBUTION / AVAILABILITY STATEMENT DISTRIBUTION STATEMENT A: APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.						
13. SUPPLEMENTARY NOTES This work is declared a work of the U.S. Government and is not subject to copyright protection in the United States.						
14. ABSTRACT This work is a case-study in the security of Z-Wave, a proprietary Internet of Things (IoT) wireless substrate, integrating sensors and actuators to provide home and office automation services. While the services minimize user burden in managing applications such as security monitoring and smart-energy, they introduce a cyber-physical attack surface into the deployed environment. Because Z-Wave is proprietary, the typical consumer is unable to ascertain the security risks in installing Z-Wave devices. To increase consumer awareness, a multifaceted security evaluation is performed on the Z-Wave transceiver system on chip (SoC). While Z-Wave devices originate from many vendors, a common transceiver facilitates interconnectivity. Herein, the transceiver is assessed as an embedded system and a communication protocol stack. Prior to a security assessment, the protocol, firmware, and non-volatile memory are partially reverse engineered to lift the veil of "security by obscurity", revealing several security concerns. One example is a key extraction attack, wherein network security is compromised by extracting cryptography keys from devices lacking physical security. In another example, several discovered network protocol vulnerabilities are combined to demonstrate a Black Hole attack, where routed Z-Wave commands are silently dropped.						
15. SUBJECT TERMS Z-Wave, Embedded Systems Security, Network Security, Home Automation, Internet of Things						
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES 216	19a. NAME OF RESPONSIBLE PERSON Dr. Scott R. Graham, AFIT/ENG	
a. REPORT U	b. ABSTRACT U	c. THIS PAGE U			19b. TELEPHONE NUMBER (include area code) (937) 255-3636, x4581; scott.graham@afit.edu	