Digital Investigation 17 (2016) 14–27

Contents lists available at ScienceDirect

Digital Investigation

journal homepage: www.elsevier.com/locate/diin

Extraction and analysis of non-volatile memory of the ZW0301 module, a Z-Wave transceiver

Christopher W. Badenhop [a], Benjamin W. Ramsey [a, *], Barry E. Mullins [a], Logan O. Mailloux [b]

[a] Department of Electrical and Computer Engineering, Air Force Institute of Technology, WPAFB, USA
[b] Department of Systems Engineering and Management, Air Force Institute of Technology, WPAFB, USA

ARTICLE INFO

Article history:
Received 4 January 2016
Received in revised form 23 February 2016
Accepted 25 February 2016

Keywords:
Z-Wave
Wireless
Embedded systems
Internet of Things forensic analysis
IoT forensics

ABSTRACT

Z-Wave is an implementation of home automation, under the broad category of Internet of Things (IoT). To date, the ability to perform forensic investigations on Z-Wave devices has largely been ignored; however, the placement of these devices in homes and industrial facilities makes them valuable assets for the investigation of criminal and adversarial actors. Z-Wave devices consist of sensors and actuators, which can be connected to the Internet via a gateway. Therefore, their memory contents may contain sensor reports of criminal activity or, more indirectly, provide evidence that the devices have been manipulated to achieve physical or cyber access. This paper provides details on extracting and programming the Flash and EEPROM memory of the ZW0301, which is a common Z-Wave transceiver module found on many Z-Wave devices. Specifically, the memory usage is characterized and several artifacts are identified. The feasibility of conducting a firmware modification attack on the ZW0301 is also explored. The results of this work identify several data structures including the node protocol information table and node adjacency table. The compiler and coding language used for the firmware image are also fingerprinted.

Published by Elsevier Ltd.

Introduction

The Internet of Things (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 (Rohan, 2015) 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. 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 connected through a gateway node to the Internet. This provides connectivity to smartphones, giving the user the

* Corresponding author.
E-mail address: bramsey@afit.edu (B.W. Ramsey).

http://dx.doi.org/10.1016/j.diin.2016.02.002
1742-2876/Published by Elsevier Ltd.

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 (FortrezZ, 2015).

Z-Wave

Z-Wave is an implementation of a complete IoT substrate, containing well-defined communication, networking, and application layer protocols. 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 ITU (2012), 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 (Hightower, 2015), 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 Non-Disclosure Agreement (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 (SigmaDesigns, 2015). The majority of all Z-Wave capability is contained on a small postage stamp sized System-On-Chip (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 required 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 (OpenZWave, 2015b).

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 (Fouladi and Ghanoun, 2013).

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 Badenhop et al. (2015).

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 Fuller and Ramsey (2015), 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 Stuttgen et al. (2015). While the article focuses on firmware modification of subsystems in desktop computers, the same principles apply to embedded system firmware. In Cui et al. (2013), 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 Picod (2014), 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 (Goodspeed, 2015). Picod et al. also provide a Software Defined Radio (SDR) implementation of a Z-Wave transceiver (Picod et al., 2014).

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 (Boztas et al., 2015), where the authors provide several memory extraction techniques analogous to Z-Wave devices. A PlayStation 4 is analyzed for digital artifacts in Davies et al. (2015).

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 (OpenZWave, 2015b). Under the Github group AFITWiSec (AFITWiSec, 2015), several Z-Wave tools are provided. These include a Wireshark dissector for Z-Wave MAC and network layers, used in Badenhop et al. (2015), 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 Bihl et al. (2015) and Patel and Ramsey (2015). 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 Yan et al. (2014). In Neisse et al. (2015), 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 paper 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 Systems. Consequently, all information provided within this paper is derived from the results of this research effort or as cited.

The remainder of this paper is organized as follows. Section system under analysis: ZW0301 provides background material for the internals of a Z-Wave device. Section Z-Wave firmware Modification attacks demonstrates that a firmware modification attack is possible for Z-Wave devices. Section Methodology imparts the extraction and analysis tools and techniques used for this effort. The analysis results of the collected firmware images are detailed in Section Results of Flash memory analysis. Section Results of External EEPROM Analysis identifies the data structures and artifacts discovered in the acquired Electronically Erasable/Programmable Read-Only Memory (EEPROM) images. The paper ends with the conclusion and future work in Section Conclusion.

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 Fig. 1. 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) (Zensys, 2007). The ZW0301 non-volatile memory

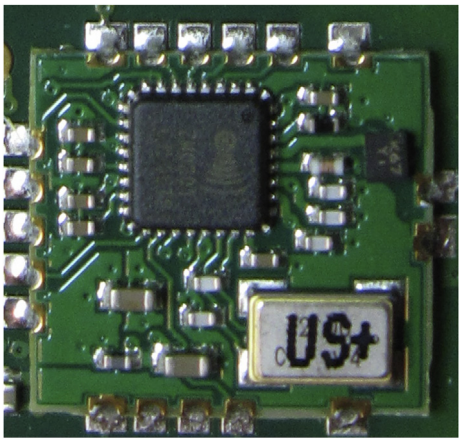

Fig. 1. ZM3102 module with ZW0301 Z-Wave transceiver, RF front-end circuit, and RF filter.

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 (Zensys, 2007).

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 (Zensys, 2007). Because of the 8051 core, the MPU is a 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 (Steiner, 2005), 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.

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 (Steiner, 2005). 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) (Steiner, 2005). The remaining SFRs are available for 8051 manufacturers to provide additional capabilities beyond the original design.

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 (Steiner, 2005).

The Intel 8051 shares its instruction set with the 8052 MPU (Steiner, 2005), 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 (Jarno, 2015) and the Keil C51 compiler (Keil, 2015b) for Windows. The assembly language for the 8051 is defined in (MetaLink, 1996).

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 (Zensys, 2007). The 32 KB Flash memory can be reprogrammed externally using an Serial Peripheral Interface (SPI) bus when the ZW0301 is placed into reprogramming mode (Zensys, 2007). While not found in all Z-Wave devices, an external 8-pin Small-Outline Integrated Circuit (SOIC) EEP-ROM 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 (Zensys, 2007). 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 (OpenZWave, 2015a) of OpenZWave source code.

Z-Wave firmware modification attacks

In Section IoT vulnerabilities, 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.

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.

The firmware modification is accomplished by first retrieving the firmware from the device using the GoodFET, shown in Fig. 2, using the process described in Section Flash memory extraction. The location of the main function is found at address 0x0649 using the approach revealed in Section INIT MX.A51 variable initialization macro. Fig. 3 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.

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

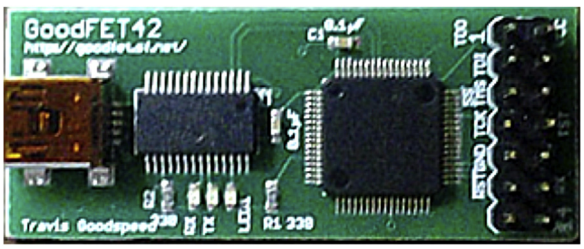

Fig. 2. GoodFET 42 board used for ZW0301 reprogramming.

```
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
```

Fig. 3. Disassembly of the main function in the firmware of an Ecolink PIR sensor.

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 Fig. 4, 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 compare a file to the firmware memory on the device, which is used to verify that the Flash memory is updated correctly. Fig. 5 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

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

Fig. 4. Disassembly of the modified main function in an Ecolink PIR sensor firmware image.

```
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.
```

Fig. 5. Verification that the firmware of the Ecolink PIR sensor is modified when compared to the factory firmware image.

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.

Methodology

From Section System under analysis: ZW0301, 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 (Motorola, 2003), 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 (Zensys, 2007), 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.

Fig. 6 is a block diagram, derived from (Goodspeed, 2015), 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.

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 package, which is capable of reading, validating, and writing the ZW0301 Flash memory. Fig. 7 shows how the ZW0301 Flash memory is retrieved using this tool.

External EEPROM extraction

Dump-Zprom

The ZW0301 provides a serial API. A tool is developed to use the FUNC_ID_ZW_READ_MEMORY serial API call (OpenZWave, 2015a) to retrieve memory from the ZW0301. 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.

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. Fig. 8 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

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. Fig. 9 shows a MiniPRO, DIP8 converter, and a Micron 25LC256 EEPROM, which is desoldered from a LynxTouch module.

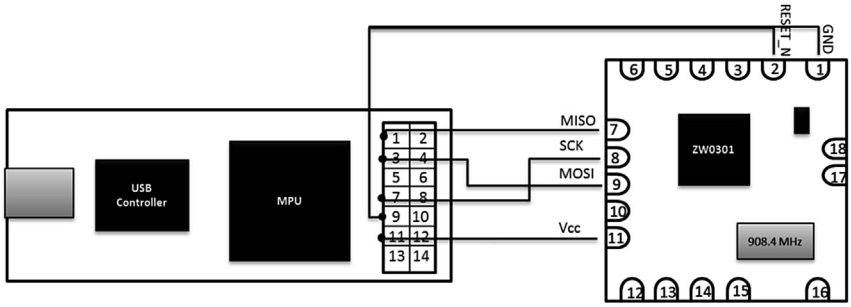

Fig. 6. GoodFET to ZM3102 interface block diagram.

```
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.
```

Fig. 7. Sample output of the goodfet.zensys info and dump commands.

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. Fig. 10 shows a portion of the memory image extracted from a 25LC256 Microchip 256 Kbit EEPROM (Microchip, 2005), which is removed from a 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. 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

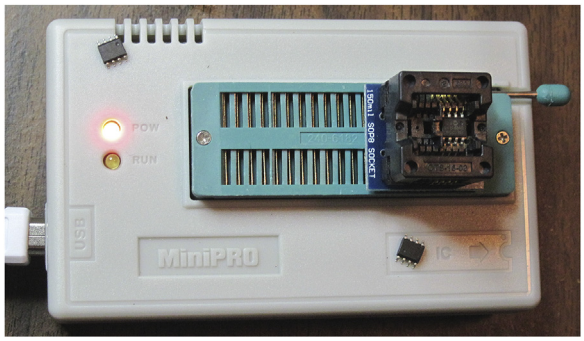

Fig. 9. MiniPRO EEPROM programmer with SOIC8 to DIP8 converter reading a Micron 25LC256 EEPROM.

controller, and LynxTouch add-on Z-Wave module are acquired using the GoodFET tool. These devices are shown in Fig. 11. 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 disassember 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

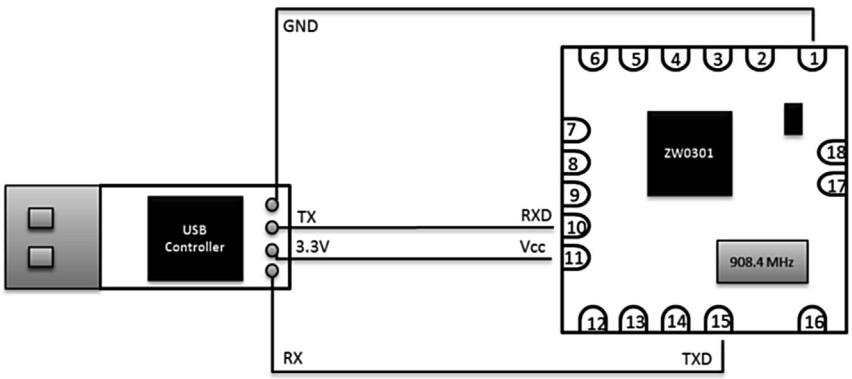

Fig. 8. USB controller to ZM3102 interface block diagram.

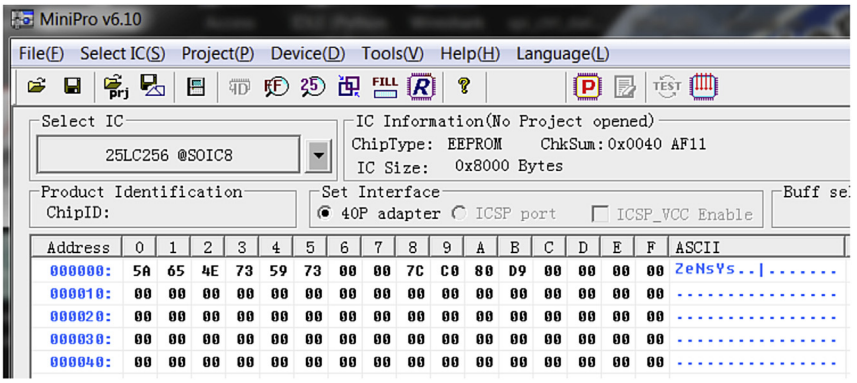

Fig. 10. MiniPRO software displaying the contents of a Micron 25LC256 EEPROM.

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 segments (Steiner, 2005). Given that IDA Pro does not know the correct numbers of handlers, it disassembles the lower memory region incorrectly, requiring manual intervention. Worse, 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 (Intel, 1982). 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 Fig. 12. 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.

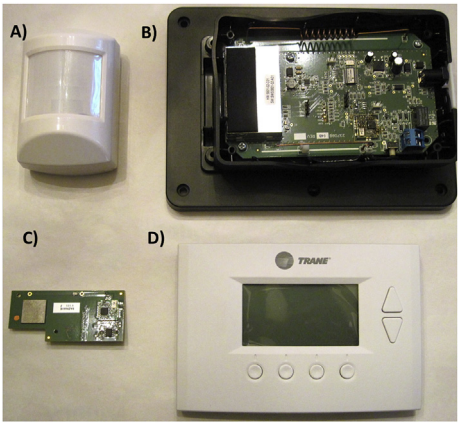

Fig. 11. Z-Wave devices with successful Flash memory exflitration – A) Ecolink PIR sensor, B) Linear garage door controller, C) Honeywell LynxTouch module, and D) Trane thermostat.

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.

To further clarify the firmware organization, Fig. 13 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

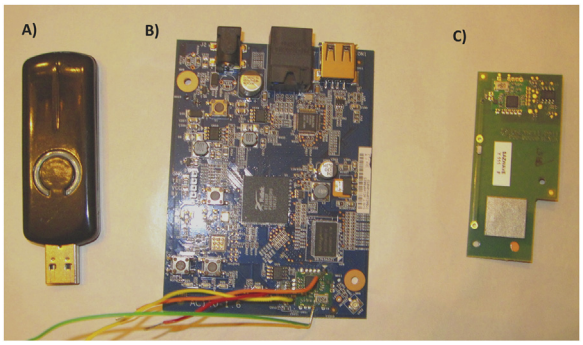

Fig. 12. Controllers used in study – A) Aeon Labs Z-Stick2, C) Vera-E controller, and C) Honeywell LynxTouch module.

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

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 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 (Picod, 2014). The remaining memory is free and is shown in white in Fig. 13.

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. Fig. 14 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 Fig. 14 contain the tokens '%s' and '%u', which are placeholders for a string and unsigned integer respectively. The existence of these 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 μVision 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 provided in the LIB directory of the C51 folder. These files are named INIT.A51, INIT_MX.A51, and INIT_TNY.A51 (Keil, 2015a). 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.

The variable initialization macro also labels the C_INITSEC segment, which is one of the first data segments embedded in the code segment. From (Keil, 2015a), this section contains the initial values of all static and global variables. Moreover, metadata 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

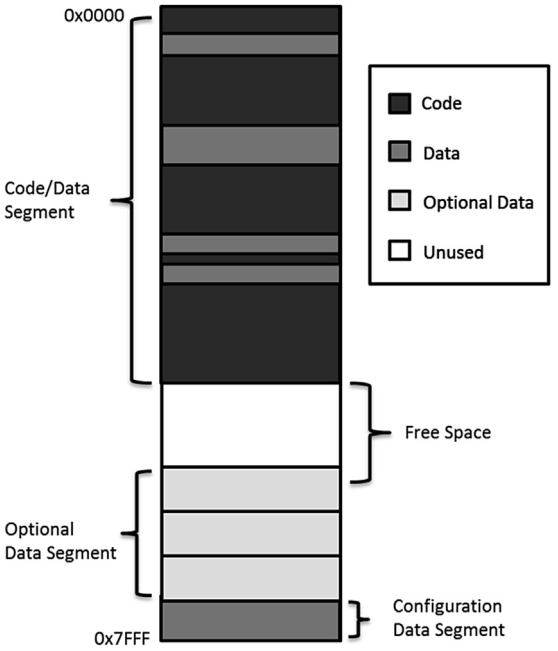

Fig. 13. ZW0301 firmware image layout.

```
0000860: 300d 0025 7353 423d 2562 6400 413d 3235  0.[%sSB=%bd].A=25
0000870: 3520 5350 003d 2562 7500 413d 3235 3520  5 SP.=%bu.A=255
0000880: 4d3d 0041 3d32 3535 2046 3d00 2573 5343  M=.A=255 F=.%sSC
0000890: 3d25 6275 0025 7353 423d 2562 7500 2573  =%bu.%sSB=%bu.%s
00008a0: 5356 2562 753d 2562 7500 2573 5254 463d  SV%bu=%bu.%sRTF=
00008b0: 2575 0025 734d 5446 3d25 7500 2573 5254  %u.%sMTF=%u.%sRT
00008c0: 433d 2575 0025 7352 5448 3d25 7500 2573  C=%u.[%sRTH=%u].%s
00008d0: 4d54 4d3d 2575 0025 7353 4525 6275 2f25  MTM=%u.%sSE%bu/%
00008e0: 6275 3d00 2530 3262 7525 3032 6275 2530  bu=.%02bu%02bu%0
00008f0: 3262 7525 3032 6275 0041 3d32 3535 2052  2bu%02bu.A=255 R
0000900: 5446 3d3f 0041 3d32 3535 204d 5446 3d3f  TF=?.A=255 MTF=?
0000910: 0041 3d32 3535 2052 5443 3d3f 0041 3d32  .A=255 RTC=?.A=2
```

Fig. 14. Printf format strings in Trane ZW0301 Flash memory.

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. Fig. 15 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.

```
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#
```

Fig. 15. Program readINITSEC decoding the C_INITSEC of the Ecolink PIR sensor.

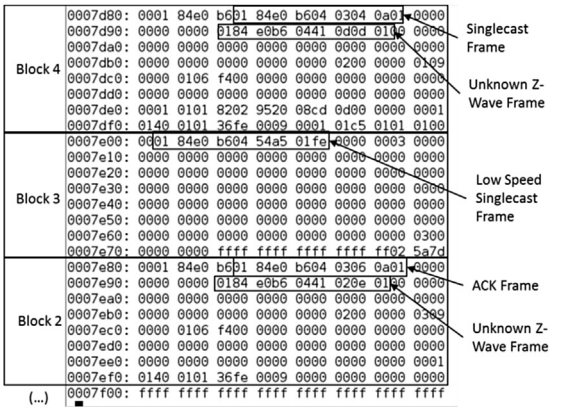

Fig. 16. Three 128-byte data blocks found in an Ecolink PIR sensor firmware image.

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.

Fig. 16 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 (ITU, 2012), the Z-Wave frame header contains at least the following nine bytes:

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

From Fig. 16, 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 (ITU, 2012), 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 (Larson, 2014), a rootkit is implemented to covertly run on a water valve Programmable Logic Controller (PLC).

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.

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. Fig. 17 shows an example byte string retrieved from an EEPROM, where the Zensys string and Home ID are

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

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 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 *Driver*::*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.

Fig. 18 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 (www.rfwirelessworld.com, 2015). An adjacency implies that two nodes are able to directly communicate. Including the controller, the largest possible Z-Wave network has 232 nodes (www.vesternet.com, 2015). 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.

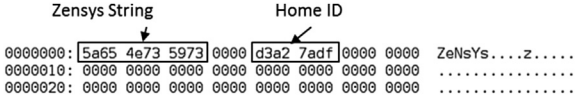

Fig. 17. Example boundary string.

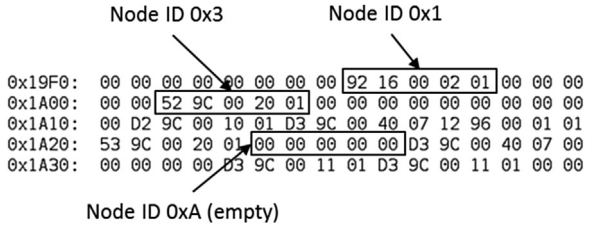

Fig. 18. Example protocol information table.

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, Fig. 19 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 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.

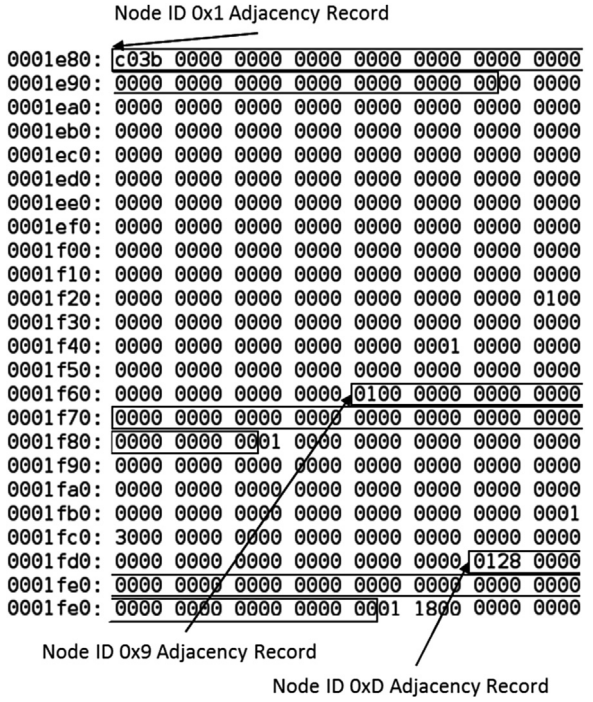

Fig. 19. Node adjacency table from an Aeon Z-Stick2 EEPROM.

Fig. 19 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 Fig. 19 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

Fig. 20 shows the controller information block at address 0x2C00 for the LynxTouch 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.

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 Fig. 21. 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 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

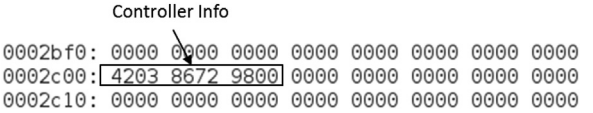

Fig. 20. Controller information within a Honeywell LynxTouch module.

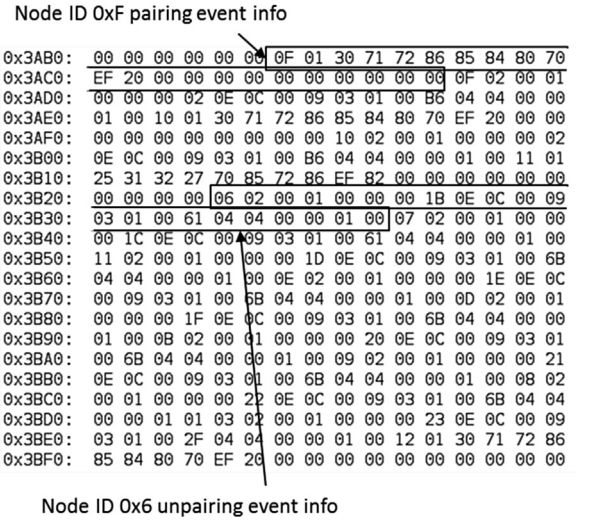

Fig. 21. Portion of a Z-Stick2 event table.

water valve user manual (FortrezZ, 2015) 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 Fig. 22. 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.

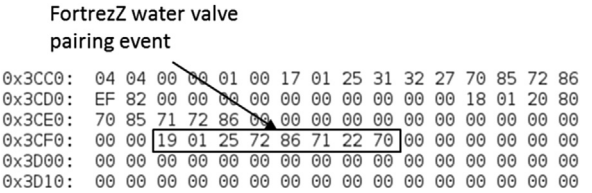

Fig. 22. Event for FortrezZ water valve paired to a Z-Stick2.

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 paper 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 (AFITWiSec, 2015).

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.

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.

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.

The views expressed in this article are those of the authors and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the United States Government. This article is approved for public release under case number 88ABW-2015-5931.

References

AFITWiSec. Afitwisec github. 2015. https://github.com/AFITWiSec [accessed 10.02.16].

Badenhop C, Fuller J, Hall J, Ramsey B, Rice M. Evaluating ITU-T G.9959: wireless systems in the critical infrastructure. In: Butts J, Shenoi S, editors. Critical Infrastructure Protection IX, IFIPS WG 11.10, vol. 9; 2015. p. 61–79.

Bihl T, Bauer K, Temple M, Ramsey B. Dimensional reduction analysis for physical layer device fingerprints with application to Zigbee and Z-Wave devices. In: MILCOM; 2015. p. 360–5.

Boztas A, Riethoven A, Roeloffs M. Smart TV forensics: digital traces on televisions. Digit Investig March 2015;12(S1):S72–80. Elsevier.

Cui A, Costello M, Stolfo S. When firmware modifications attack: a case study of embedded exploitation. In: NDSS Symposium; April 2013.

Davies M, Read H, Xynos C, Sutherland I. Forensic analylsis of a Sony PlayStation 4: a first look. Digit Investig March 2015;12(S1):S81–9. Elsevier.

FortrezZ. WV-01 Wireless Z-Wave Water valve Owner's manual and installation guide. FortrezZ; 2015.

Fouladi B, Ghanoun S. Security evalution of the Z-Wave wireless protocol. In: Presented at Blackhat USA; July 2013.

Fuller J, Ramsey B. Rogue Z-Wave controllers: a persistent attack channel. In: SenseApp. IEEE; 2015.

Goodspeed T. GoodFET homepage. December 2015. http://www.goodfet.sourceforge.net [accessed 15.12.15].

Hightower D. Sigma designs acquires Zensys holdings. December 2015. http://mwrf.com/content/sigma-designs-acquires-zensys-holdings [accessed 12.10.15].

Intel. External product specification for the MCS-51 object module format. Intel; September 1982. Tech. rep.

ITU. ITU recommendation G.9959: short range narrow-band digital radiocommunication transceivers – PHY and MAC layer specifications. Tech. rep. Telecommunication Standardization Sector of ITU; 2012.

Jarno A. SDCC – Small Device C Compiler. October 2015. http://sdcc.sourceforge.net/ [accessed 28.10.15].

Keil. Cx51 user's guide: variable initialization code. November 2015. http://www.keil.com/support/man/docs/c51/c51_ap_init.htm [accessed 18.11.15].

Keil. Keil C51 C compiler. October 2015. http://www.keil.com/c51/c51.asp [accessed 28.10.15].

Larson J. Miniaturization. In: Presented at Blackhat USA; August 2014.

MetaLink. 8051 cross assembler user's manual. MetaLink Corporation; January 1996.

Microchip. 25Aa256/25LC256 256 K SPI bus Serial EEPROM. Microchip; 2005.

Motorola. SPI block guide. 3rd ed. Motorola; February 2003.

Neisse R, S G, Fovino I, Baldini G. Seckit: a model-based security toolkit for the internet of things. Comput Secur June 2015:1–17. Elsevier.

OpenZWave. Openzwave Defs.h. November 2015. http://www.openzwave.com/dev/Defs_8h_source.html [accessed 18.11.15].

OpenZWave. Openzwave homepage. October 2015. http://www.openzwave.com/home [accessed 28.10.15].

Patel J, Ramsey B. Comparison of parametric and non-parametric statistical features for Z-Wave fingerprinting. MILCOM 2015:378–82.

Picod J. Dumping firmware out of a z-wave asic. February 2014. http://blog.cassidiancybersecurity.com/post/2014/02/Dumping-firmware-from-ASIC [accessed 10.02.16].

Picod J, Lebrun A, Demay J. Bringing software defined radio to the penetration testing community. In: Presented at Blackhat USA; August 2014.

Rohan S. Home automation and control market worth $12.81 billion by 2020. November 2015. http://www.marketsandmarkets.com/PressReleases/home-automation-control-systems.asp [accessed 18.11.15].

SigmaDesigns. Dev kits - Z-Wave - Sigma designs. November 2015. http://z-wave.sigmadesigns.com/dev_kits#z-wave_whats_in_kit [accessed 18.11.15].

Steiner C. The 8051/8052 microcontroller. Boca Raton, FL USA: Universal Publishers; 2005.

Stuttgen J, Vomel S, D M. Acquisition and analysis of compromised firmware using memory forensics. Digit Investig March 2015;12(S1):S50–60. Elsevier.

www.rfwirelessworld.com. z-wave protocol stack — z-wave protocol layer basics. December 2015. http://www.rfwireless-world.com/Tutorials/z-wave-protocol-stack.html [accessed 15.12.15].
www.vesternet.com. Understanding z-wave networks, nodes & devices. December 2015. http://www.vesternet.com/resources/technology-indepth/understanding-z-wave-networks [accessed 15.12.15].
Yan Z, Zhang P, Vasilakos A. A survey on trust management for internet of things. Netw Comput Appl 2014;42:120–34. Elsevier.
Zensys. ZM3102N Z-Wave module datasheet. Zensys; October 2007.