FACULTY OF INFORMATION TECHNOLOGY CTU IN PRAGUE

ASSIGNMENT OF MASTER'S THESIS

Title:	Honeypot for wireless IoT networks
Student:	Bc. Simon Štefunko
Supervisor:	Ing. Tomáš Čejka, Ph.D.
Study Programme:	Informatics
Study Branch:	Computer Systems and Networks
Department:	Department of Computer Systems
Validity:	Until the end of summer semester 2019/20

Instructions

Study the state-of-the-art of hardware and software tools for receiving, processing, and transmitting signals of wireless protocols for Internet of Things (IoT).
Focus on possibilities how to use commonly available hardware devices and Software Defined Radio (SDR) technology.
Based on discussion with the supervisor, choose a wireless protocol that will be analyzed in details in this thesis.
Design a prototype of an "IoT honeypot" using the SDR; this honeypot will allow to monitor, store, and answer to the wireless communication like any other legitimate device.
Implement the IoT honeypot and test it with the available set of IoT devices/sensors (supplied by the supervisor).
Create data sets of legitimate communication of IoT devices using the developed device.

References

Will be provided by the supervisor.

prof. Ing. Pavel Tvrdík, CSc.
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague October 9, 2018

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Master's thesis

Honeypot for wireless IoT networks

Bc. Simon Štefunko

Department of Digital Design
Supervisor: Ing. Tomáš Čejka, Ph.D.

May 7, 2019

Acknowledgements

At first I would like to thank my thesis supervisor Ing. Tomáš Čejka, Ph.D., Faculty of Information Technology at Czech Technical University in Prague. The door to Tomáš Čejka office was always open whenever I ran into a trouble spot or had a question about my research or writing. He steered me in the right the direction whenever he thought I needed it.

I would also like to thank my parents and my girlfriend for providing me with unfailing support throughout my years of study. This accomplishment would not have been possible without them. Thank you.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague on May 7, 2019 .....

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Štefunko, Simon. *Honeypot for wireless IoT networks.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2019.

Abstrakt

Ciele tejto práce ležia v teoretickej analýze konceptu Internet vecí (IoT) a jeho bezpečnostných problémov, praktickom výskume a vývoji nového unikátneho zariadenia zvaného "IoT honeypot". Analytická časť práce sumarizuje existujúce hardvérové a softvérové riešenia, a sústredí sa na technológiu Softvérom definovaného rádia (SDR), ktorá bola použitá na vývoj IoT honeypot-u. Vyvíjaný prototyp v súčasnosti podporuje rozšírený Z-Wave protokol. Avšak, dizajn je dosť univerzálny na to, aby v budúcnosti podporoval ďalšie IoT protokoly. Motiváciou tejto práce bolo vytvoriť zariadenie, ktoré dokáže zbierať informácie o IoT komunikácii, detegovať potenciálnych útočníkov, a pôsobiť ako návnada, ktorá komplikuje útočníkom objaviť a prebrať kontrolu nad skutočnými nasadenými IoT zariadeniami, ako sú senzory, spínače, a podobne. Výstupom tejto práce je funkčný IoT honeypot, ktorý podporuje viacero režimov fungovania (napríklad pasívny alebo interaktívny režim), a môže byť nasadený ako súčasť Z-Wave infraštruktúry. Predstavuje komplement k ostatným bezpečnostným nástrojom a mechanizmom, ktoré zvyšujú úroveň bezpečnosti IoT infraštruktúry.

Klíčová slova IoT, SDR, honeypot, Z-Wave

Abstract

The goals of this thesis lay among theoretical analysis of the Internet of Things (IoT) concept and its security issues, and practical research and development of a new unique device called "IoT honeypot." The analytical part of the thesis summarizes existing hardware and software solutions and concentrates on Software Defined Radio (SDR) technology, which was used for the development of IoT honeypot. The developed prototype currently supports a wide-spread Z-Wave protocol. However, the design is universal enough to support other IoT protocols in the future. The motivation of this thesis was to create a device that can collect information about IoT traffic, detect potential attackers, and act as a decoy that complicates attackers to discover and hack real deployed IoT devices, such as sensors, switches, and so on. The result of the thesis is a working IoT honeypot that supports multiple modes of operation (such as passive or interactive mode), and that can be deployed as a part of a Z-Wave infrastructure. It is as a complement to other security tools and mechanisms that increase the security of IoT infrastructure.

Keywords IoT, SDR, honeypot, Z-Wave

Contents

Introduction **1**

1 State-of-the-art **3**
1.1 Internet of Things 3

2 Analysis **9**
2.1 Available tools for the IoT 9
2.2 Software Defined Radio 12
2.3 (IP) Honeypots 21
2.4 Requirements Analysis 23
2.5 Selected Components 25

3 Design of Proposed Work **27**
3.1 System architecture 27
3.2 Modes 29
3.3 Virtual Decoys 31
3.4 Commands 32
3.5 Data structures 33
3.6 Detection method 35

4 Realization **39**
4.1 Control 39
4.2 Logging 46
4.3 Persistence of data structures 46
4.4 Interaction 47
4.5 Validation 48
4.6 Data Set of Communication 48

5 Testing **51**
5.1 Test Environment 51

5.2 Testing Scenarios 51
5.3 Test results 53
5.4 Replication 56
5.5 Summary 57

Conclusion **59**
Future Work 60

Bibliography **61**

A Acronyms **67**

B User Manual **69**
B.1 Installation 69
B.2 Usage 70

C Contents of enclosed CD **71**

List of Figures

1.1 Three-layer architecture of the IoT 4

2.1 General frame structure 12
2.2 General MAC frame format 12
2.3 Z-Wave frames on a waterfall display of GQRX 14
2.4 Reception of Z-Wave frames using rtl-zwave 15
2.5 Preparing and transmission of Z-Wave frames using waving-z . . . 16
2.6 GRC blocks of Scapy-radio for reception of Z-Wave frames 18
2.7 GRC blocks of Scapy-radio for transmission of Z-Wave frames . . . 19
2.8 Sniffed frames transmitted by POPP dimmer 20
2.9 Exploitation of POPP Dimmer 20
2.10 Standard usage of a honeypot 21

3.1 The IoT honeypot architecture 28
3.2 A conflict of node indentifiers 30
3.3 Real and Virtual Z-Wave networks 31
3.4 Two phases of an internal replication process of virtual decoys . . . 33
3.5 Configuration class 34
3.6 Processes and shared information 35
3.7 Attack detection of the IoT honeypot 37

4.1 Blocks of variables 41
4.2 Recording communication and creating decoys 43
4.3 Two Pipes used for communication 43
4.4 Malicious modified frame detected 44
4.5 Malicious duplicate of one of last ten frames sent by the honeypot 44
4.6 External replication of decoys from Network A to Network B . . . 45
4.7 Interaction between an attacker and a Z-Wave outlet 48
4.8 Real network consisting of a door sensor, a controller and a dimmer 48

List of Tables

2.1 Data rate, modulation and coding 12
2.2 Comparison of SDR hardware 13

5.1 Detection of 100 malicious frames prepared by waving-z 53
5.2 Detection of 100 malicious frames transmitted by scapy-radio . . . 54
5.3 Detection of 100 malicious frames transmitted by scapy-radio . . . 54
5.4 Detection of 20 malicious frames transmitted by scapy-radio with interaction 55
5.5 Detection of 20 malicious frames transmitted by scapy-radio with interaction and real Z-Wave network 55
5.6 Detection of 20 malicious frames transmitted by scapy-radio with interaction and real Z-Wave network 56
5.7 Responding to 20 malicious frames transmitted by scapy-radio with interaction and real Z-Wave network 56

Introduction

The Internet of Things could be described by many definitions. Most of them describe the Internet of Things as a concept of connecting everyday objects to a network using embedded devices. The main reason for this approach is to build smart systems composed of information-gathering nodes, usually called sensors, and nodes which can to interact with the environment. This could lead to smarter adaptation based on data gained from the real world. The phrase "Internet of Things" was likely first mentioned in 1999 by a British technology pioneer Kevin Ashton, as he explained in [1].

The total number of connected IoT devices has been estimated at 7 billion in 2018. The Internet of Things market has recently experienced rapid acceleration. We expect an increase in the number of devices to 10 billion by 2020 and 22 bilion by 2025, as IoT Analytics GmbH has predicted in [2].

There are several ways of building an IoT network. Individual IoT networks may vary in their architecture, topology, hardware components, way of communication, software application and usage. However, some design features are common to most architectures. A node usually has a micro-controller unit, a communication unit, one or several sensors or actuators and a power supply.

The IoT nodes need to be able to perform modulation and demodulation of signals to achieve wireless communication. For research, development, and experiments, it is useful to use some universal hardware components and software solution to set up modulation and demodulation pipeline for signal processing. This concept is usually called Software Defined Radio (SDR). According to [3] the SDR is a system, which provides these functions using a software and a few hardware components. A universal tool like this may provide functionality usable in IoT networks, but it also brings new ways to compromise security.

This thesis aims to create a honeypot device for IoT network. The term honeypot means it is a system or a part of a system that looks seemingly vulnerable to a potential attacker, but the main purpose for this decoy is to

capture attacks, log information about them and entice a potential hacker. A honeypot will also make the attackers think twice about executing their attack, as it may be confusing and scary for the hacker. One of many ways to increase the security of IoT is to use an IoT honeypot that is attractive enough for the attacker and representative of the IoT context, as has been described in [4].

The main goals of this thesis are focused on analyzing and improving IoT security. It should never be underestimated, because the successful attacks may cause devastating consequences in every sphere of our lives nowadays and especially in the future. Our mission is to ensure technologies to be reliable enough. Civilization without that is not prepared for such a great impact as the IoT will bring. After all, building IoT networks and interconnecting everyday objects means a partial transfer of control over our lives to technologies. Capturing the traffic of a popular smart home IoT protocol may lead to its analyzing and anomaly detection.

One of the main results of this thesis is a prototype of an IoT honeypot, which is able to perform such activity. Besides capturing, it should also fulfill other functionality of a honeypot – being attractive for an attacker. Penetration tests of a selected smart home protocol should also provide information about the availability of all components used for attack, poor security standards and the evidence of a threat. This thesis deal with IoT security mainly. Flaws of popular IoT protocols for home automation open many ways how to compromise its security. IoT honeypot is one of a few approaches to ensure detection and prevention against real attacks and therefore it should be used primarily for vulnerable technologies available in the real world and the market. However, there is a lack of usage of this type of security element and this thesis discusses the design of IoT honeypot for a popular home automation protocol.

Main goals of this thesis are the analysis of existing threats that IoT brought, vulnerabilities of the Z-Wave protocol and usage of the SDR technology within smart devices. The output of this thesis is also the implementation of the IoT honeypot, which can be used for luring attackers and threat detection on Z-Wave networks. Chpt. 1 describes available standard software and hardware tools that support the IoT concept. Analysis of the protocol Z-Wave, SDR tools usable for penetration testing and existing IoT honeypots is described in Chpt. 2. Chpt. 3 includes the design of the architecture of the IoT honeypot, description of its main components, methods and functions. Chpt. 4 explains implementation of the IoT honeypot according to its design. This chapter also summarizes various tools, which were used for implementation. Tests scenarios and results are summarized in Chpt. 5. The conclusion in Chpt. 5.5 discuss achieved results and further potential improvements.

Chapter 1

State-of-the-art

1.1 Internet of Things

The term "Internet of Things" was first mentioned publicly by Kevin Ashton in 1999. He explained his perception of this concept later in [1] in 2009. He considered all data on the Internet to be information and ideas cumulated by people. His vision was to empower computers to gather information about the real world on their own. Sensors could help computers to understand and observe the real world without the limitations of human-entered data." Kevin also believes that the Internet of Things will be able to bring changes comparable to the scale of changes the Internet itself has brought.

1.1.1 Description

Several definitions of the term "Internet of Things" can be found. Most of them describe the IoT as an idea of connecting everyday objects to the Internet. Such activity has multiple purposes. Sensors are used for a smarter and precise gathering of information. Simply, It is better to get as much relevant information as possible. Conversion of information from analog to digital allows applying appropriate data processing, which may lead to a reaction. The Internet of Things does not include only sensor nodes, but also actuators. Thus its abilities exceed the perception and allow influencing the real world.

The architecture of the IoT according to [5] consists of three main layers. The bottom layer includes all kinds of information gathering nodes and actuators. This kind of devices provides an ability to the IoT to perceive the real world. The main purpose of the bottom layer is to simply gather as much relevant information as possible although many end nodes are devices with low power consumption. A lot of them use batteries as a power supply. It is quite usual for a sensor node to wake up once three hours, perform some measuring activities and send data through messages to upper layers. There are several technologies that allow node collaboration in local and short-range

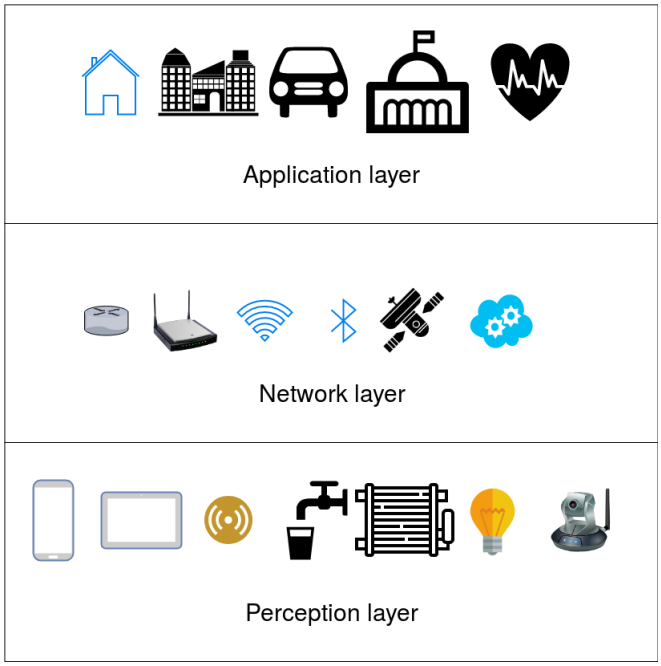

Figure 1.1: Three-layer architecture of the IoT [5]

networks. This layer is usually called Perception layer.

The middle layer is responsible for the bidirectional transmission of messages. According to [6], the major components of this layer are all kinds of connections, network management systems, and the Internet. There are typically transceivers, gateways, wired and wireless connections and common network elements. The IoT concept brings a lot of new use cases that would be absurd without wireless communication. For example, a simple network of nodes that are measuring soil moisture on a field does not seem to be as a good idea with a wire interconnecting some gateway with all the nodes. Thus there is the possibility of usage of electromagnetic waves to transmit data. This layer is called Network layer and provides heterogeneous systems mainly for transmission, but also for example for data filtering and data aggregation. Well-known technologies like Wi-Fi, LTE, 3G, Bluetooth, Zigbee [7], which are compared in [5], cloud computing platforms with an example in [8], LoRaWAN [9] and Z-Wave [10] are indispensable parts of the Network layer of the IoT nowadays.

Data processing is mainly executed in the top layer of this simple view of an IoT architecture. However, there are no boundaries that would require compliance with this architecture. Devices with a built-in user interface or systems, where some functionality of a server is decentralized between multiple gateways using [11], remains. This layer, according to [6] usually called Application layer, should guarantee data authenticity, data integrity, and data

confidentiality. Common application protocols allow interprocess communication using ports. Some popular application layer protocols are for example MQTT, HTTP, web socket and AMQP.

Trying to map IoT protocols on existing architecture models like the OSI model would become pointless. Also, the IoT has brought many new applications and smart use cases, but with many constraints. For example, devices using wireless communication usually supplied by batteries are expected to use protocols that ensure low-power computational complexity. Another example is a short range of radio waves used during wireless communication. All these constraints require protocols that provide efficient management of these conditions. Communication protocols in the context of the IoT may be divided into two major groups.

LPWAN stands for low power wide area networks. Sigfox [12], LoRaWAN [9] and Cellular [13] are main communication protocols in the context of the IoT, which are suitable for a distance of dozens of kilometers. A comparison can be found in [14]. Several use cases in [15] teach us, that usage of these technologies are reasonable. One packet per day is a common message sending frequency for smart metering of water, gas or electricity. However, this information is valuable, accessible from long distances and it is not necessary to read it from built-in meters anymore. A lot of energy would be saved using smart street lighting and frequency would be a few packets per night. There are many more use cases, like smart parking or vehicle fleet tracking.

Due to physical and legislative constraints, we need to use multiple ways of communication [16]. That is the reason, why IoT networks with short-range exist, even if we could use LPWAN. One of the most popular short-range protocols is 6loWPAN [17]. This protocol connects low power IP driven nodes to cloud systems and works with IPv6 packets carried in small link layer frames. Jonas Olsson has explained in [17] IP technology may be considered as its major advantage. ZigBee is also a popular protocol in a group of short-range IoT protocols. ZigBee is described in [18] as a protocol that may be used for low cost personal high-level networks for a short range and for data transmission for longer distances at the same time. It is suitable for applications, where the long battery life and low data rate are required. Besides star topology, ZigBee can also use a mesh and a tree topology of a network. Essential protocol for the IoT is BLE. Unlike classic Bluetooth, BLE is intended for low power usage. A low latency ensured by this protocol may be an advantage in many IoT applications. Kevin Ashton has explained in [1] that RFID has also included in initial ideas of the IoT: “RFID and sensor technology enable computers to observe, identify and understand the world - without the limitations of human-entered data.” RFID [19] provide a simple tagging using unique information programmed to tags for shopping, health care, security, agriculture and many more.

The concept of the Internet of Things is well-known nowadays. There are many ways of achieving benefits from this concept. Many various hard-

ware tools, software applications, and protocols have been designed thanks to bright predictions about the IoT. IoT networks usually consist of many various devices. The architecture of nodes should be optimized for its purpose. However, there still are some common design features preserved in all IoT devices. A typical IoT node consists of one or several processors, sensors or actuators, a communication module, and a power supply. Nodes typically communicate with some more complex gateways or with each other. Selection of devices depends on the application. The market offers a wide range of devices. Unfortunately, there are several factors that can cause design flaws, as was explained in [15].

1.1.2 Hardware and software

The number of IoT providers is growing. Nowadays it is much easier to build an IoT application. Everything the regular user has to do is to build or buy an end device which can be connected to the provider's network. The market offers ready-made nodes for easy usage, which provides special functionality. Using these nodes usually doesn't require special knowledge of protocols and programming. On the other hand, the possibility of building something from scratch remains. Such a decision likely brings more flexibility and responsibility. Of course, it depends on the design. Buying a popular micro-controller with a prepared communication module can be still considered easy to use. However, implementing, for example, a custom protocol may become an unnecessarily time-consuming activity.

It is difficult to predict the relationship between market and research, but according to [20] standardization for IoT is expected. However, the term "Internet of Things" is still considered equivocal. The ambiguity is caused by differences among multiple architectures and perspectives. The variability of IoT allowed in [21] software solutions to be assembled into multiple classes. The classes could be sorted for example by a level of complexity of a software solution. The lowest class represents devices at the end of the network, which have a built-in user interface. Conversely, devices of the highest class can also provide independent conduct without human interaction. IoT solutions can be distinguished from the lowest software layers. Some of them consist only of software libraries to ensure the unencrypted communication, and the other include sophisticated repeating mechanisms and encryption. Some of them are platform-dependent, and the others are not.

End devices most commonly use wireless communication. There are many types of wireless communication, and all have one feature in common - transmission of information through the air by using electromagnetic waves without requiring any wires. SDR is a technology, which replaces hardware components with software blocks, as was explained in [3]. These software blocks are responsible for modulation and demodulation of a signal. How it processes a

signal is a question of configuration, so SDR brings more flexibility to wireless communication.

Network attacks are reaching a new level, and according to IoT predictions, the impact of a successful attack on the IoT network would be devastating. A probability of a wireless attack and its availability increases thanks to technologies such as the SDR rapidly. An example may be found in [22]. There are multiple ways to protect our devices and detect any suspicious activity. One of these ways is the usage of a honeypot. Generally, a honeypot is a system or a part of a system, which is used as a decoy. A honeypot may also represent a trap also in a context of IoT networks, and there are several solutions of IoT honeypot [23, 24, 25].

Chapter 2

Analysis

This chapter includes analysis of available tools for the IoT, the protocol Z-Wave, usage of the SDR technology for penetration testing and existing solutions of the IoT honeypot element.

2.1 Available tools for the IoT

Current technologies allow us to connect to IoT networks in several ways. The most simple way is to buy prepared products from certain manufacturers, do the pairing process and minimal configuration of applications. Usage of a micro-controller may be suitable for a more advanced user. The market offers ready-made communication modules, so call shields, which can be just connected to an appropriate micro-controller. The complexity of this end-device is up to the user. User can build a simple sensor reporting its values periodically, or smart, complex actuator with multiple sensors with a custom protocol. The other side of communication, usually gateway or controller, is necessary of course. There are several ways how to build an application. The standard procedure is to send all the data to the cloud because a steep increase in devices is expected in the future. An example of the whole system may be the usage of Arduino Uno [26] with multiple sensors and a LoRa Dragino Shield [27] as an end-device, using a LoRa gateway [28] and a LoRa server [29] able to handle the communication. Usage of LoRa@FIIT [11, 30] and STIOT [31, 32] protocols may be another example with a functional decentralization from a server to multiple gateways.

There are many IoT providers, who offer connectivity using various technologies and cloud. Paying a reasonable amount of money to a provider may be an efficient way, how to satisfy needs for IoT network without buying more expensive hardware for gateways. Everything, what user needs nowadays may be only a sensor node able to measure and transmit information to providers gateway. Cloud and web pages prepared for efficient usage allows simple handling with data. Bright prediction [2, 33] about the growing number

of connected IoT devices also brings many questions about the final impact of this change to our lives. Letting smart devices and systems think for us and control our world for us requires trust, which may be achieved by quality security at every layer. Wireless communication allows almost everyone with suitable hardware to try to sniff the communication. Thanks to encryption, identification and secure key exchange are ensured the main needs of security. However, flaws have been found on many protocols and technologies after the mass selling of unsecured products. The attacks should be detected since the beginning – at the lowest layer.

2.1.1 Z-Wave

The concept of a smart home is based on metering using sensors and remote controlling of everyday objects. Besides these main activities, collecting data for analysis and more efficient automation has great expectations. A smart home needs protocols. Protocols usually define rules between interconnected devices, which ensure proper communication using messages. Z-Wave, created by the Danish startup Zensys [34], became one of the most popular protocols used for home automation. However, they can act as repeaters during the active state. Slaves supplied by batteries are not able to listen continually, so controllers calculate routes without including them. There are several types of slave nodes, but generally, the Slave node type receives frames and replies if necessary. An example is a power outlet. Other roles of controllers and slaves are defined as explained in [35].

Z-Wave is a protocol implemented by multiple manufacturers. The market offers a wide range of products with Z-Wave technology. There are several ways how to automate your household. Not only variability of end-devices is high, but also the selection of controller or gateway is a matter of decision. More and more homes are getting home automation nowadays. Using a Z-Stick Gen5 [36] to control a POPP wall plug dimmer [37] is a simple example of home automation. However, the software is also needed. OpenZWave [38] is a free software library that interfaces with selected Z-Wave PC controllers. It allows to create applications suitable for simple manipulation and responding to devices on a Z-Wave network. One of these applications can be Domoticz [39], which offers nice user interface in browser to control IoT devices at your household.

Every Z-Wave network is separated by the Home ID [35, pg. 6], which is a 32 bits unique identifier. The manufacturing process of all controllers includes pre-programming these identifiers. Resetting a controller causes generating a new random Home ID. All additional nodes assign the same Home ID, during the installation of the Z-Wave network. The initial controller remains as the Primary Controller. Identification of nodes in a network is ensured by the 8 bit Node ID [35, pg. 6]. This value is assigned to a node by a Primary Controller or an Inclusion Controller, and it is unique within a network.

Z-Wave devices for home automation are usually plug-and-play. Thus its deployment can be very easy. The main reason is to satisfy user convenience because it is one of the major factors influencing the market of IoT devices. However, this approach may bring security vulnerabilities. First, some devices [36] event didn't support basic encryption, and it makes unwanted listening of Z-Wave frames very easy. Besides listening, taking control over devices is possible and in many cases may cause huge damage.

The first generation security solution based on Security Command Class is described in [40]. Security Command Class, also called S0, provides end-to-end security on the application level. Encryption is ensured by AES [41] symmetric block cipher algorithm using a 128 bit key length. Even though S0 provides data freshness, confidentiality, and message integrity, the security layer does not protect against denial of service attacks, hardware side-channel attacks, traffic analysis, protocol-side channel attack or attacks against application-layer security. More detailed information about the S0 may be found in [40]. However, the network key transmitted during the pairing process of S0 between the nodes was using a key of all zeroes. This vulnerability was published and explained in [42]. Shortly, the network key could be sniffed by an attacker within range.

According to [43], the S2 Security is best-in-class security while maintaining the user-friendliness and power efficiency and may be considered as the first true smart home security solution. The S0 has been superseded by S2. There are explained ways how to protect your Z-Wave network at multiple layers, for example, usage of the home control gateway to establish a secure connection via its LAN interface to a trusted portal server on the Internet. Like the S0, the S2 uses AES-128 encryption and combination with S2 authentication and Nonce scrambling [43], and there is no known way to break this protection. However, network key distribution is a classic chicken-and-egg problem. Diffie-Hellman key-exchange [44] solves this problem and also a problem of the weak S0 security. The key-exchange can also involve authentication by the entry of a 5 digit code into the controller. Even though the key should not be possible to intercept when is S2 used, the possibility of a successful attack remains. Maintaining backward compatibility between S2 and S0 devices enables downgrade attack during the pairing process. The attack was explained in [45].

Z-Wave protocol has multiple layers. One of them is PHY layer, also called the physical layer, which is complying with ITU-T Recommendation G.9959 [46] and therefore Z-Wave devices can communicate on multiple sets of communication parameters listed in Table 2.1. Z-Wave devices can also operate on multiple frequencies. The frequencies are dependent on the region and table of coverage can be found in [47].

The transmission and reception of PHY protocol data units (PPDUs) are enabled thanks to the PHY data service. The MAC protocol data unit (MPDU) is passed to the PHY layer as a PHY service data unit (PSDU). In

Table 2.1: Data rate, modulation and coding [46]

Data Rate	Bit Rate	Modulation	Coding
R1	9.6 kbit/s	FSK	Manchester
R2	40 kbit/s	FSK	NRZ
R3	100 kbit/s	GFSK	NRZ

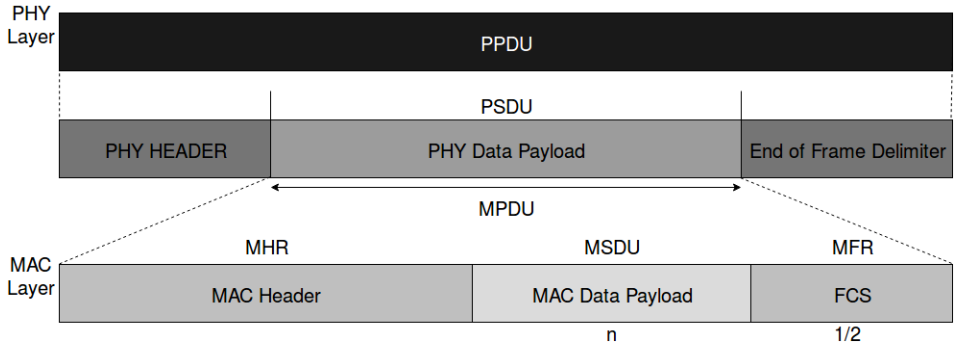

Figure 2.1: General frame structure [46]

most cases, the MAC layer forward frames to the higher layers for processing. The general frame structure is illustrated in Fig. 2.1.

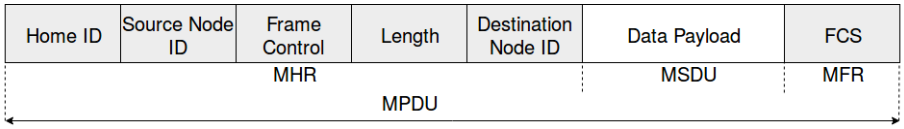

Figure 2.2: General MAC frame format [46]

The MAC service data unit (MSDU) contains payload data from the network layer. MAC header (MHR) contains HomeID, source node ID, MAC frame control field, a frame length field, and destination node ID. The MFR include a non-correcting frame check sequence (FCS). The MHR, MSDU, and MFR together form the MAC protocol data unit (MPDU), as explained in [46]. The example of the general MAC frame is illustrated in Fig. 2.2. Data payload usually includes command classes, commands, and values specified in [48].

2.2 Software Defined Radio

Software Defined Radio (SDR) is a technology that will shape the future of wireless communication. The main purpose of the SDR is to ensure soft-

Table 2.2: Comparison of SDR hardware [54, 52, 53, 55, 51]

Product	RTL-SDR	Airspy	HackRF	USRP B200
Radio Spectrum	25 MHz - 1750 MHz	24 MHz - 1750 MHz	30 MHz - 6 MHz	50 MHz - 6 GHz
Bandwidth	3.2 MHz	10 MHz	20 MHz	61.44 MHz
TX	No	No	Yes	Yes
Duplex	-	-	Half	Full
Dynamic Range	50 dB	80 dB	48 dB	70 dB
Sample Rate	2.56 MHz	10 MHz	20 MHz	61.44 MHz
Approx. price	$25	$199/$249	$299	$745

ware computation of existing hardware components usable for digital signal processing (DSP). The concept of SDR is not new, but fast-evolving digital electronics allows to render more and more processes. Joseph Mitola, its creator, has described in [49] an ideal software radio transceiver as a system with digital-to-analog converter (DAC) and analog-to-digital converter (ADC) at the antenna and at handset allow all radio transmit, receive, signal generation, modulation and demodulation, timing, control, coding, and decoding functions to be performed in software. However, the SDR, of course, includes many non-DSP hardware components like anti-aliasing filters or power handling. Besides the military, research and industrial usage the SDR is used by radio amateurs nowadays.

RTL-SDR dongles [50] are very popular among radio amateurs thanks to their low price. Users usually use these dongles for receiving radio or Digital Video Broadcasting-Terrestrial (DVB-T) signal. Airspy [51] offers more sensitive signal reception thanks to wider frequency range, and its natural high dynamic range allows for excellent signal noise ratio (SNR) and reception of weak signals when in the presence of nearby strong signals. However, none of these tools is capable of transmission. HackRF One [52] is an SDR device capable of transmission and reception of radio signals from 1 MHz to 6 GHz. The USRP B200 [53] should provide more sensitive reception thanks to higher a sample rate. A more detailed comparison of some popular SDR hardware products is in Fig. 2.2, where parameters like frequency range of radio spectrum, bandwidth, the ability of transmission, duplex, dynamic range ensuring reception of very weak or very strong signals, sample rate, and approximate prices are compared. Most of these devices use Universal Serial Bus (USB).

The list of SDR supported software is long. All software for RTL-SDR can be found in [56]. GQRX [57] is open-source software for an SDR receiver which runs on Linux and Mac systems. GQRX provides a waterfall display, standard Fast Fourier transformation (FFT) spectrum, and filter settings. GQRX offers a graphic interface usable for searching a signal and analyzing the spectrum,

and therefore Z-Wave frames can be displayed in Fig. 2.3. Another popular and flexible tool is a GNU Radio [58], which is described in Sec. 2.2.2.

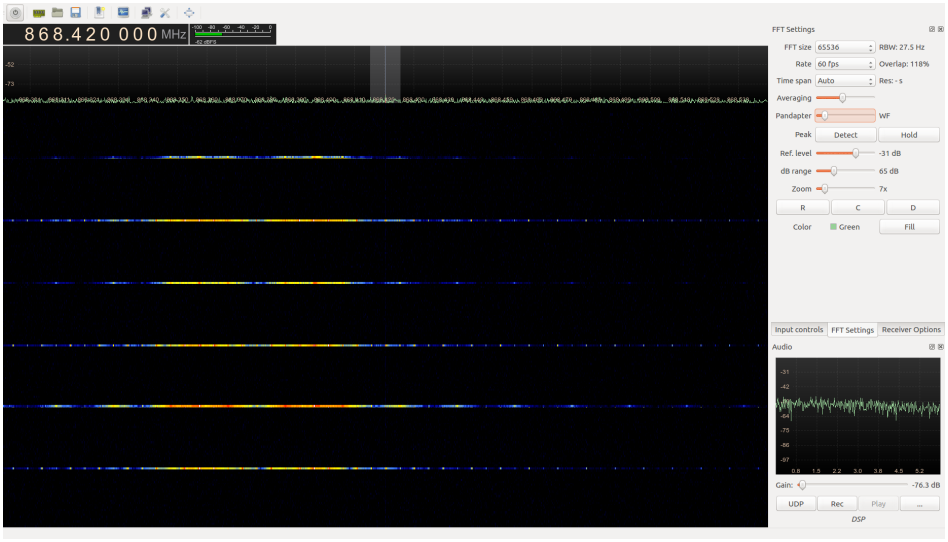

Figure 2.3: Z-Wave frames on a waterfall display of GQRX

2.2.1 Penetration tests

Thanks to the majority of wireless IoT protocols, the SDR became a useful tool not only for security evaluation but also for undesirable activities. Frames of wireless protocols are transmitted through the air and therefore not only verified devices can receive them. Notably, in the case of unencrypted communication, the danger of adverse eavesdropping becomes much more realistic. Penetration testing is a method of testing networks, computer systems or applications executing harmless attacks to find its security vulnerabilities exploitable by an attacker. Several tools can be used for effective penetration tests for Z-Wave protocol also.

The software library librtlsdr [59] offers several tools and one of them is rtl-sdr. The program rtl-sdr tests the existence of RTL2832, a chip used in most SDR dongles, and performs data transfer functions. Code base of rtl-sdr includes a Frequency Modulation (FM) receiver. Its command line interface offers options to interact with hardware, like setting a frequency, a sample rate, a gain or just to specify where I/Q data from output will be dumped. However, receiving signal is not enough to be able to work with ITU G.9959 frames, which Z-Wave protocol use. A received signal needs to be demodulated and decoded. A preamble needs to be detected, as it indicates the start of a frame. Even though a demodulator and a decoder are main components, filters and clock-recovery are usually used for improvement of reception. Decoded data

can be parsed into frames. Transmitting is preceded by a reversed process, that includes encoding, modulation and adding a preamble.

Z-Wave signal demodulation and decoding can be done by rtl-zwave [60]. A pipeline is used for transferring a received signal from rtl-sdr program to rtl-zwave. The decoder of the rtl-zwave supports FSK and GFSK demodulation. The demodulator supports Manchester decoding and NRZ decoding. All data rates of Z-Wave are implemented too, so all sets of Z-Wave PHY parameters in Tab. 2.1 are supported. An example of reception is Fig. 2.4. A reversed process ensuring modulation, coding, and transmitting of frames is not supported in rtl-zwave.

```
$rtl_sdr -f 868.42e6 -s 2048000 -g 25  - |./rtl_zwave
Found 1 device(s):
  0:  Realtek, RTL2838UHIDIR, SN: 00000001

Using device 0: Generic RTL2832U OEM
Found Rafael Micro R820T tuner
[R82XX] PLL not locked!
Sampling at 2048000 S/s.
Tuned to 868420000 Hz.
Tuner gain set to 25.40 dB.
Reading samples in async mode...
df11f6300841030d012503009700
Frame num 1 ends on sample 6755059 Fofs=-66413.771352 SR=9130.628622
df11f6300841050d0125030091
Frame num 2 ends on sample 7296452 Fofs=-74103.808753 SR=37961.075070
df11f6300841050d0125030091
Frame num 3 ends on sample 7352690 Fofs=-74738.474151 SR=38102.325581
df11f6300841050d0125030091
Frame num 4 ends on sample 7528944 Fofs=-74828.778693 SR=38173.345760
df11f6300841060d0125030092
Frame num 5 ends on sample 7578514 Fofs=-74094.666595 SR=38066.914498
df11f6300841060d0125030092
Frame num 6 ends on sample 7672276 Fofs=-74326.853883 SR=38031.569174
df11f6300841060d0125030092
Frame num 7 ends on sample 7766535 Fofs=-74209.206671 SR=38031.569174
```

Figure 2.4: Reception of Z-Wave frames using rtl-zwave

The program waving-z [61] can be used not only for the reception but also for transmitting. Usage of reception is very much like rtl-zwave, but the analysis proved that waving-z can demodulate and decode only frames using FSK demodulation and NRZ decoding on 40 Kbps data rate. Transmitting must be preceded by modulation and encoding hexadecimal frames into files. Its content can be later transmitted, using the HackRF One, as shown in Fig. 2.5.

2.2.2 GNU Radio

GNU Radio is an open-source development kit that provides signal processing blocks to implement a software radio. It is widely used in academia, research, industry, government and among radio amateurs. A major advantage of GNU Radio is reusable easy-to-use blocks for DSP. These blocks usually implement standard algorithms, but scalability remains. Creating flow graphs consisting of blocks for DSP is done in GNU Radio Companion (GRC). It is a graphical

```
$./wave-out -p 'd6 b2 62 08       01   41  03  0d     07          25   01 ff' >
 turn_on_device_7.cs8
d6 b2 62 08 01 41 03 0d 07 25 01 ff 63
[x] HomeId: d6b26208, SourceNodeId: 1, FC0: 41, FC1: 3, FC[speed=0 low_power=0 a
ck_request=1 header_type=1 beaming_info=0 seq=3], Length: 13, DestNodeId: 7, Com
mandClass: 25, Payload: 01 ff
$hackrf_transfer -f 868420000 -s 2000000 -t turn_on_device_7.cs8
call hackrf_sample_rate_set(2000000 Hz/2.000 MHz)
call hackrf_baseband_filter_bandwidth_set(1750000 Hz/1.750 MHz)
call hackrf_set_freq(868420000 Hz/868.420 MHz)
Stop with Ctrl-C
 3.9 MiB / 1.000 sec =  3.9 MiB/second
 0.3 MiB / 1.000 sec =  0.3 MiB/second

User cancel, exiting...
Total time: 2.00034 s
hackrf_stop_tx() done
hackrf_close() done
hackrf_exit() done
fclose(fd) done
exit
```

Figure 2.5: Preparing and transmission of Z-Wave frames using waving-z

tool that simplifies the process of implementing software radio interconnecting blocks to flow graphs to achieve required signal processing. Otherwise, users would have to implement a code of blocks using Python and to change parameters of blocks like the frequency or the sample rate in the GRC influence the generated Python code. However, there is still a possibility to modify the code. Sometimes, a user needs blocks, which are not in the original version of the GRC. For example, user needs to process signal with Frequency-Shift Keying (FSK) demodulation [62] and Manchester decoding [63]. The block ensuring Manchester encoding is not implemented in the original set of blocks. This block can be implemented using Python or C++ and used in the graphical interface of GRC. Besides the official documentation of GRC blocks including examples and tutorials, the communities interested in GNU Radio can be found on several web pages and social networks. GRC blocks can be used to receive and transmit Z-Wave frames.

Scapy-radio [64] has been described in [22]. It consists of GNU Radio Blocks and Scapy [65]. Communication between them is ensured by a UDP socket. Scapy-radio includes GNU Radio Blocks specialized for reception and transmitting of frames of several IoT protocols. Z-Wave frames are demodulated and decoded by blocks for the reception and sent as a UDP packet to a socket. Scapy-radio can parse these frames into standardized network data structures used by Scapy. Beside demodulation and decoding, other blocks like filters and clock-recovery sustain better signal reception. First block ensuring receiving signal from SDR hardware can be replaced by a block for another compatible SDR hardware. Authors of Scapy-radio implemented several blocks and *Zwave PacketSink* is one of them. It decodes a demodulated signal using NRZ encoding. A signal can be displayed by several sink blocks, which are included in GRC. Scapy-radio GRC blocks for the reception of Z-Wave frames are shown in Fig. 2.6. Scapy-radio offers reception blocks only for

the data rate 40 Kbps, FSK modulation and NRZ encoding. EZ-Wave [66] is a project, which offers few tools for evaluating and exploiting Z-Wave networks using SDR. This tool uses Scapy-radio for reception and transmission.

The blocks of Scapy-radio for transmitting Z-Wave frames are described in [67]. Modified received frames or new forged frames to be transmitted are sent to a socket, which represents UDP server. The *Preamble* block adds a preamble to frames. Length of PHY frames is calculated by the block *PDU to Tagged Stream*, which ensures conversion to a bitstream too. The block *Not* is needed because the block *Gaussian Frequency Shift Keying (GFSK) Mod* uses opposite symbol mapping as ITU-T Recommendation G.9959 frames. GFSK modulation with a parameter *BT=1* effectively minimizes a response of Gaussian filter leaving a standard FSK rectangular pulse shape. GRC does not include an FSK modulation block. The desired sample rate 20 MHz, which is the maximum sample rate for HackRF One and *osmocon Sink*, is achieved thanks to re-sampling a signal at a rate of 250:1. This rate is calculated using:

$$\frac{40Ksymbols/sec \times 2samples/symbol}{20Msamples/sec} = \frac{1}{250}$$

Osmocon Sink is the last block, which ensures transmitting through HackRF One. The sink and SDR hardware can be replaced. Blocks are sequentially linked as show in Fig. 2.7.

2.2.3 Exploiting Z-Wave device

SDR allows to receive and transmit Z-Wave frames. The modular architecture of the Scapy-radio framework allows to use rtl-zwave for reception of Z-Wave frames. Received frames are sent packed in UPD packet to a socket used by Scapy-radio and parsed. Scapy-radio blocks for transmitting remains and can be used for transmission of sniffed, modified or forged frames.

Once an unencrypted frame is sniffed, *Home ID*, *Source ID*, *Destination ID* and more information can be extracted. Especially, the *Home ID* and *Source ID* are important for executing simple exploitation of a Z-Wave device and taking control over it. Sniffed frames can be displayed thanks to Scapy-radio in the terminal, as shown in Fig. 2.8. A device, which includes command class *SWITCH_BINARY* [48] reports in these frames the change of a state - turning ON (0xff) and OFF (0x00).

In addition to displaying sniffed frames, Scapy-radio supports effective manipulation with them. Creating an attack frame starts with duplication one of sniffed frames. The *Source ID* needs to be set to the controller's ID (1), and *Destination ID* is set to the ID of a target device. A sequence number should be updated. A command class must be set to the *SET* class. A payload includes a command for the target device turning ON (0xff) or OFF (0x00). Recalculation of the CRC should be a final step before transmission. Transmission of the frame can be affected by variable TX gain, which is one

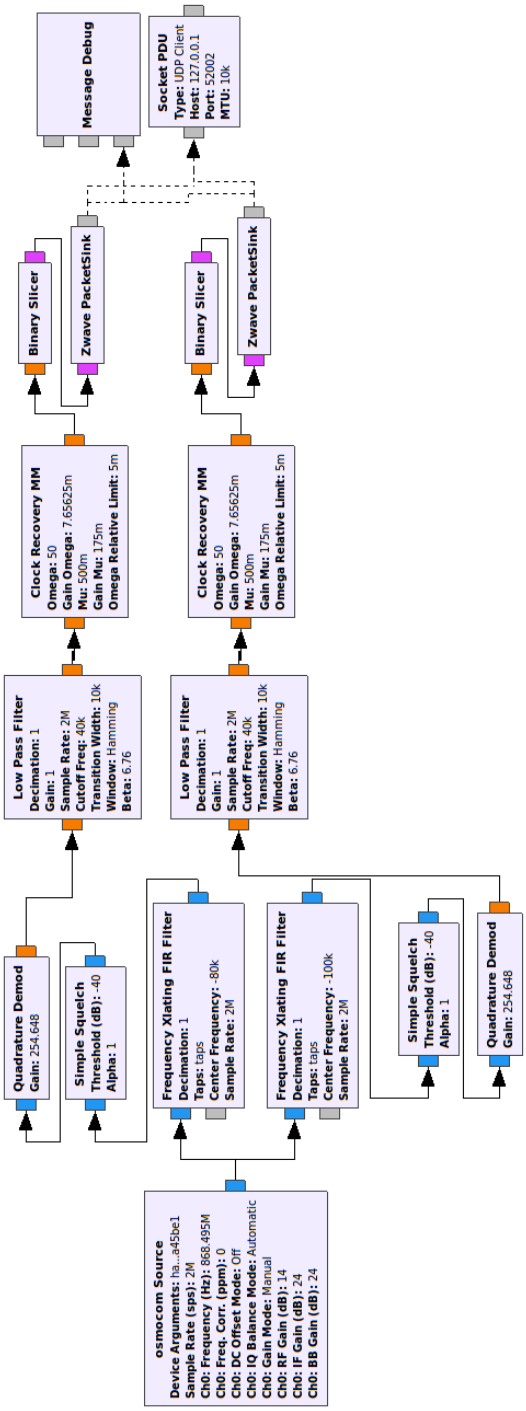

Figure 2.6: GRC blocks of Scapy-radio for reception of Z-Wave frames

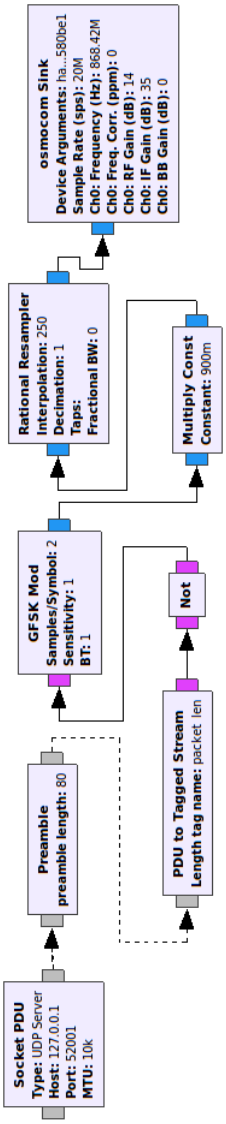

Figure 2.7: GRC blocks of Scapy-radio for transmission of Z-Wave frames

of the significant factors influencing a range of a signal. In case of a frame sniffed directly from a controller, only sequence number, command and CRC needs to be changed, as shown in Fig. 2.9.

Another attack can be executed by active scanning. Z-Wave includes frames, which demand a device's state. Transmitting a large number of these frames and incrementing their *Home ID* can lead to exposure of unencrypted Z-Wave network. The *Home ID* is 32 bits long, so there are more than 4 billions of combinations. This active scanning was tested on a range of 1000 values of *Home ID*, where the *Home ID* of target network was the last one.

```
###[ Gnuradio header ]###
  proto     = 1
  reserved1 = 0x0
  reserved2 = 0
###[ ZWaveReq ]###
     homeid    = 0xdf11f630
     src       = 0x8
     routed    = 0L
     ackreq    = 1L
     lowpower  = 0L
     speedmodified= 0L
     headertype= 1L
     reserved_1= 0L
     beam_control= 0L
     reserved_2= 0L
     seqn      = 7L
     length    = 0xd
     dst       = 0x1
     cmd_class = SWITCH_BINARY
     crc       = 0x93
###[ ZWaveSwitchBin ]###
        cmd       = REPORT
###[ Raw ]###
           load      = '\x00'
```

```
###[ Gnuradio header ]###
  proto     = 1
  reserved1 = 0x0
  reserved2 = 0
###[ ZWaveReq ]###
     homeid    = 0xdf11f630
     src       = 0x8
     routed    = 0L
     ackreq    = 1L
     lowpower  = 0L
     speedmodified= 0L
     headertype= 1L
     reserved_1= 0L
     beam_control= 0L
     reserved_2= 0L
     seqn      = 14L
     length    = 0xd
     dst       = 0x1
     cmd_class = SWITCH_BINARY
     crc       = 0x65
###[ ZWaveSwitchBin ]###
        cmd       = REPORT
###[ Raw ]###
           load      = '\xff'
```

Figure 2.8: Sniffed frames transmitted by POPP dimmer [37]

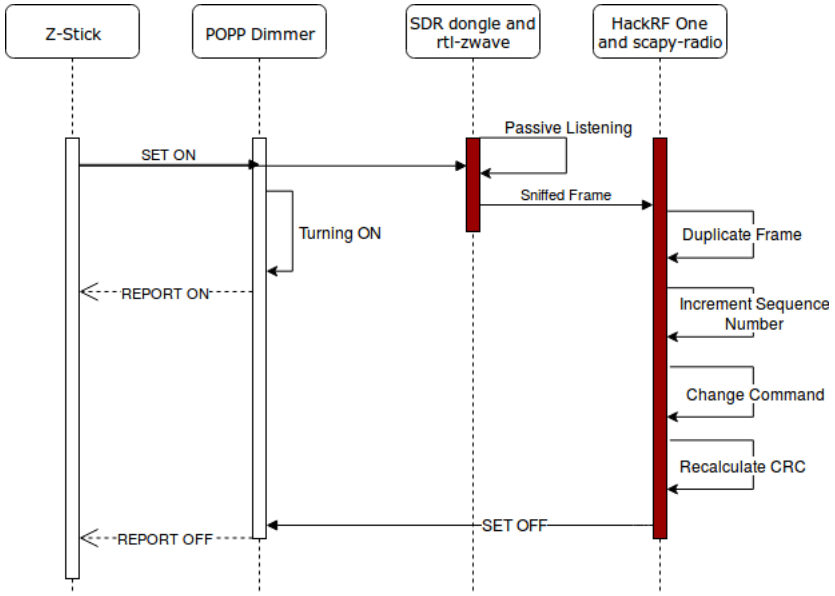

Figure 2.9: Exploitation of POPP Dimmer

Transmission of 1000 frames and successful answer reception approximately lasted 1.96 seconds using HackRF One. The speed of transmission can be changed, however too high speed causes a frame is not detected by the device. The device used for this attack was POPP Dimmer. A time needed for scanning the whole range of possible values for *Home ID* is calculated here:

$$\frac{2^{32}}{1000} \times 1.96\,seconds \approx 2338.37\,hours$$

Passive scanning brings more reliable results. Sniffing information about devices and using them to build frames for controlling them represents a real threat to IoT users.

2.3 (IP) Honeypots

Honeypots are rarely used security elements. The main idea of a honeypot is to seem vulnerable to lure attackers during malicious activities. It does not have any production value, so false positives, which can be found on intrusion detection systems do not exist on honeypots. Any network traffic coming from honeypot can be considered successful compromising. Honeypots can be classified into multiple classes according to several factors. Major factors like interaction level, data capture, containment or communication interface are described in [68]. Another classification is by the implementation because honeypots can be virtual or physical. There is also usually a difference between production honeypots and research honeypots. Generally, a honeypot should look as real as possible and should be attractive to a hacker. It is essential to mention, a honeypot security is not comprehensive. Mechanisms of a honeypot will never replace usually used security elements like network intrusion detection system or prevention systems, but it can extend the amount of gathered information about attacks, draw hackers away from important resources and detect attack attempts. Traditional usage with other network elements is illustrated in Fig. 2.10.

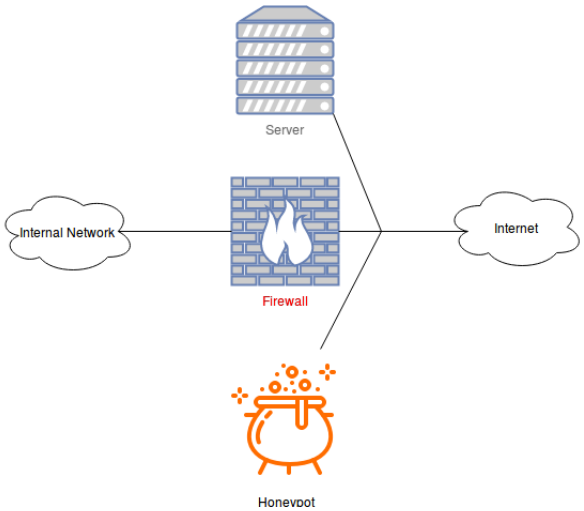

Figure 2.10: Standard usage of a honeypot

Several solutions of honeypots were developed in the past. Google Hack Honeypot [69] provided records information about time, the source IP address and the file that has been accessed on the web server by a malicious Google search. Another example is Honeyd [70], which represents the more traditional implementation of a production honeypot. It is a program, that is able to create multiple virtual honeypots on a network. Telnet, FTP, SMTP or HTTP can be simulated using the Honeyd. It monitors millions of IP addresses at the same time, and when it sees a connection attempt, it interacts with attacks. Project Honey Pot [71] is a distributed system for identifying spammers and spambots. Honeypot farms provide functionalities of a honeypot as a service. Ideas of a honeypot can be implemented in multiple ways. For example, honeynets are considered a high interaction technology in a group of honeypot projects.

Honeypots have also found their place in the IoT. The IoT communication lays in specific physical layers. Therefore a variety of IoT honeypot architectures is high. The main purpose of Telnet IoT honeypot [25] is to catch botnet binaries. This project includes a Python Telnet server, which acts as a honeypot for an IoT malware and it can automatically analyze botnet connections and map botnets and its networks. The Mirai DDoS exploited security vulnerabilities of IoT devices. ThingPot, which is described in [72], is implemented to mimic the Philips Hue smart lighting system [73] and five types of attacks against smart devices have been captured during 1.5 months thanks to this honeypot. Several solutions of IoT honeypots are described in [24, 74, 75, 76, 77], which vary in an architecture and purposes.

Captain Caleb E. Mays describes the design and construction of a honeypot to defend a building automation system in [23]. This work deals with security vulnerabilities of protocol INSTEON [78], that is used for building automation. His research also improves on his previous research using the SDR to receive and transmit INSTEON commands. Primary goals of his research are to develop methods for sending and receiving INSTEON network communication using the SDR and develop a virtual honeypot to emulate INSTEON devices. According to his work, a honeypot should be a set of targetable authentic decoys. A lone IoT light bulb is of little significance to an attacker when it is not connected to a network, so a honeypot should implement several virtual decoys. Simulated network traffic from multiple virtual devices should convince a potential attacker that every decoy is a part of a legitimate automation network. Since an attacker can interact, it is necessary to sustain more or less authenticity of decoys. Even a low interaction honeypot like this can answer to an attacker and emulate the limited functionality of devices. For example, a light bulb can either be ON or OFF. A decoy should respond by a message of the state of the light bulb to an attacker.

A most related project to this thesis is described by Captain Caleb. The output of his work is a prototype of the IoT honeypot that lures attackers using decoys, which can respond. The main difference between our proposed

IoT honeypot is a process of creating decoys. The IoT honeypot that was implemented by Captain Caleb E. Mays uses configuration files for creating virtual decoys. Communication between these decoys is also specified in another configuration file. Process of creating decoys for our proposed IoT honeypot is described in Sec.3.3. Our process, which uses the recording of legitimate communication, allows transmitting more sophisticated communication with a minimum effort.

2.4 Requirements Analysis

Several requirements of the Z-Wave IoT honeypot prototype using the SDR needs to be defined. Functional requirements represent major functionalities of the designed IoT honeypot. Non-functional requirements describe attributes of the IoT honeypot.

2.4.1 Functional Requirements

- **Monitoring**

  One of the major goals of the IoT honeypot is receiving of Z-Wave frames from real devices Z-Wave devices. The IoT honeypot should also receive frames transmitted by the SDR, like HackRF One, to ensure detection of attempts to exploit virtual decoys. Receiving unexpected frames from real Z-Wave devices can bring results in anomaly detection in case of an inability of receiving frames from the attacker.

- **Storing**

  The IoT honeypot should allow storing of gained information and log some of them. Acquired information should include information about attempts to exploit decoys and information about real Z-Wave devices to be able to monitor their activity. Attackers usually do not deal with answering acknowledgments frames to devices after an exploit. For example, the attacker may change the state of a dimmer, and a dimmer reports its status several times if it does not receive an answer. That may be an indicator of taking control over the dimmer. All gathered information should be reasonably aggregated and represented.

- **Generation of simulated traffic**

  As well as common IP honeypots, this should lure attackers to uncover their presence and plans. Since this honeypot will work on a physical layer of the IoT, the only way of building virtual decoys is to generate simulated traffic. These decoys will simulate limited functionalities of real Z-Wave devices. There is no production value of these decoys. Therefore any traffic directed to them is considered an attack attempt. Configurable sets of virtual decoys should be a part of the IoT honeypot.

- **Interaction**

  Besides creating an illusion by generated traffic of virtual decoys, a certain level of interaction should be achieved. The IoT honeypot should be able to answer to attackers at least with basic frames. The IoT honeypot should also keep information about states of decoys to answer the most legitimately.

2.4.2 Non-functional Requirements

- **Scalability**

  The IoT honeypot should allow scalability at several layers. First, usage of these multiple IoT honeypot systems should lead to cooperation, not a conflict. This can be achieved using decoys of a single network at a one honeypot system. The count of configurable decoys should be another scalable attribute of this honeypot.

- **Modular architecture**

  Modular architecture allows replacement of specific modules of a system for example in the case of incompatibility with different hardware or effective integration with other systems.

- **Configurability**

  Configurability of the IoT honeypot should be ensured by parameters and configuration files. An option of changing communication parameters should remain. Configuration files should allow to save and load saved configuration.

- **Targetability**

  Targetability is an attribute of a designed honeypot, which describes it as something attractive for attackers. Decoys of the IoT honeypot should seem to be potential targets of attacks. Virtual decoys, which could not be targeted, would make the idea of luring impossible.

- **Authenticity**

  Ensuring authenticity means allowing interaction with the virtual decoys. Targetable decoys could be quickly revealed without proving some level of interactivity. It is evident that fully implemented functionalities of virtual decoys could be time-consuming and performance-consuming and therefore limited functionality can be sufficient enough to fool the attacker for a time that is needed for detection and logging.

- **Prevention**

  Virtual decoys of the IoT honeypot extend a Z-Wave network from an attacker's point of view after sniffing communication. Virtual decoys

bring a chance, that attacker will select a virtual decoy as the target. Probability of selecting real Z-Wave devices depends on a count of virtual decoys.

- **Compatibility**

  The goal of this thesis is to implement a prototype of the IoT honeypot. It will be implemented on operating system Linux Ubuntu 16.04, to achieve compatibility with standard GNU/Linux systems and therefore ensure a possible future integration with one of the popular microcontrollers.

2.5 Selected Components

Final implementation of the IoT honeypot will use popular SDR platforms. RTL-SDR dongle will be used for a receiving of Z-Wave frames. HackRF One will allow to transmit Z-Wave frames. These components are chosen because of a price and compatibility with popular SDR software tools. Transmitting by HackRF One cause a frequency offset of frames, however regular Z-Wave devices are still able to receive these frames. Using rtl-sdr program to receive signal and rtl-zwave to demodulate and decode it allows receiving frames using all sets of communication parameters, but the frequency offset cause inability to receive frames with a frequency offset. Scapy-radio uses GRC blocks, which allows a higher level of flexibility. GRC blocks provides an option of receiving frames with the frequency offset. Even if GRC blocks misses implementation of receiving and transmitting blocks for two of three sets of communication parameters, the honeypot will be able to monitor and interact with the vast majority of Z-Wave devices. Using Scapy-radio GRC blocks is more suitable for functionalities of the IoT honeypot, because it will focused on standard communication, furthermore, the modular architecture of Scapy-radio ensure easy adding other components for DSP. Since EZ-Wave offers examples of tools, which can be implemented using Scapy-radio, EZ-Wave is the best choice, as it includes Scapy-radio too.

Chapter 3

Design of Proposed Work

This chapter deals mainly with the design of an IoT honeypot. Its design includes the description of its architecture mainly composed of a controller, a monitor, a receiver, a responder, a traffic generator, a transmitter, a way of configuration and modes of the IoT honeypot.

3.1 System architecture

The architecture of the IoT honeypot should be modular and simple. Separation of a receiver and a transmitter allows for replacement of components like hardware, modulators, demodulators, encoders, decoders, and frame parsers.

- **Controller**

  The controller processes parameters, arguments and coordinate main functionalities and modes of the IoT honeypot. The controller starts monitoring activities and generation of simulated traffic according to user's input. This component initializes most of the other parts and process configuration input from a terminal or a configuration file. Beside initializing main functionalities and processing of user's commands, the controller offers functions like displaying the status of the honeypot and resetting it to default state, which means clearing all saved records and network information.

- **Configuration**

  Configuration of the IoT honeypot includes setting communication parameters like a frequency, a sample rate of transmission gain and parameters like paths to network information and records of decoys. Individual parameters can be set using optional arguments of commands of the IoT honeypot or providing a configuration file. The configuration is provided to other components as a shared data structure.

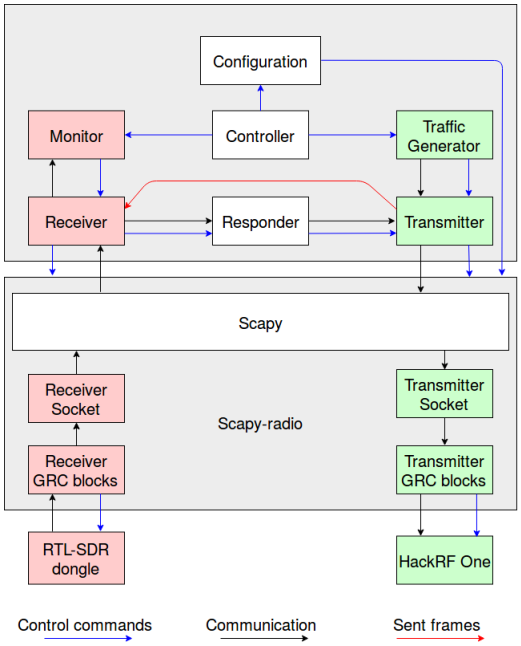

Figure 3.1: The IoT honeypot architecture [22]

- **Monitor**

  The monitor's main task is to decide what to do with malicious frames. The monitor detects and logs malicious activities. Its main task is to start, control and pass arguments for the receiver and process information, which the receiver passes to the monitor.

- **Receiver**

  Reception of Z-Wave frames is an essential activity that leads to monitoring and authentic interaction. The receiver processes frames passed by the Scapy-radio and distributes frames to the monitor in case of malicious activities and the responder according to their content. The receiver filters frames sent by the transmitter thanks to information passed from the transmitter. The receiver pass frames from attackers to the monitor, which logs attack attempts. The receiver takes commands from the monitor and sets proper receiving mode. Computing CRC of frames and mapping of real Z-Wave network is also included in the receiving process. The receiver also does recording of legitimate communication and saves it for future usage.

- **Responder**

  The receiver forwards commands and frames to the responder. Its main task is to prepare legitimate frames to emulate responses of virtual de-

coys and handle states of decoys to persuade attackers about the legitimacy of devices and to gain more time to detect and react to attack attempts. Acknowledgment frames and report frames, which are the vast majority of responds are then transmitted using the transmitter.

- **Traffic Generator** The traffic generator simulates traffic of virtual decoys. Its main task is to load recorded frames prepared for a currently configured network. In case of missing records of frames, replication of records of another network or recording of communication is needed.

- **Transmitter**

  The transmitter processes frames and prepares them to be sent. The transmitter ensures setting a sequence number and calculated CRC. A very important task of this component is to pass information about sent frames to the receiver. The receiver will be able to filter frames sent by the transmitter and recognize frames sent by attackers.

The analytic part Sec. 2.2.2 of this thesis explains and proves that RTL-SDR dongles are applicable as receivers of Z-Wave frames. The receiving components consist of a RTL-SDR dongle and software, that is compatible with the dongle. The Scapy-radio offers flexibility because GRC blocks can be changed or extended. The architecture of the Scapy-radio, which includes communication trough a socket, provides an option of replacing components for DSP. The analytic part Sec. 2.2.2 also describes the usage of the HackRF One device with the Scapy-radio.

3.2 Modes

This section describes various modes of the IoT honeypot. The IoT honeypot offers several various commands that provide main functionalities. Modes are compositions of settings or configurations that affect the functionality and can bring different results.

- **Record mode**

  Received frames can be saved into files with an extension *.pcap*, which are usually used for network analysis. Scapy-radio offers an option of installing Wireshark [79] with dissectors to recognize Z-Wave frames. Recording provided by Scapy-radio can be helpful not only for later traffic analysis but also for recording frames, which can be modified and used as traffic of virtual decoys. Transmitting is not allowed during the record mode of the IoT honeypot. The IoT honeypot records traffic, randomizes identification numbers *Node ID* in frames and saves frames for future usage. It is recommended to check if all displayed and received frames during record mode are authentic.

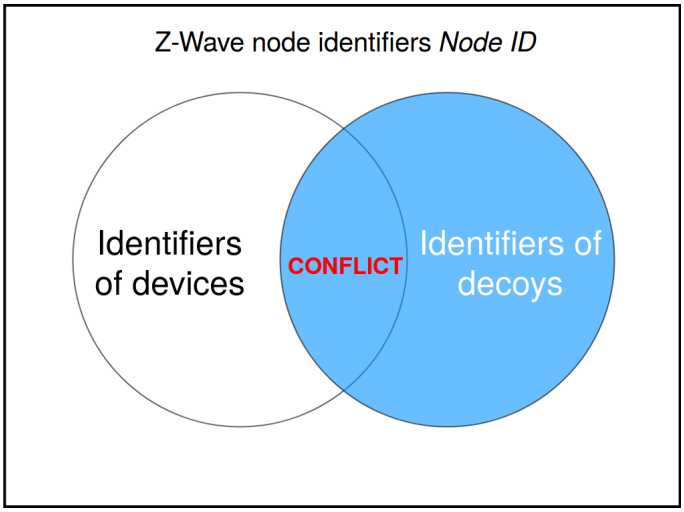

Figure 3.2: A conflict of node identifiers

Process of setting values of *Node ID* complies with rules of Z-Wave. Mapping of the real network during several modes avoid conflict between real nodes and decoys. This way of recording and modifying frames is a simple way, how authentic and targetable decoys can be created. It is recommended to record a Z-Wave network that the IoT honeypot should protect. Replication of existing records is described in Sec. 3.4 which leads to expanding a network of existing decoys or creating a brand new network of decoys.

- **Default mode**

  The default mode of the IoT honeypot includes functionalities that support essential ideas of the IoT honeypot. The IoT honeypot monitors the spectrum using GRC blocks. The receiver processes all received Z-Wave frames. Frames sent by real devices are used for mapping of real Z-Wave network. This mapping is used for setting values of *Node ID* of virtual decoys. It is recommended to map the whole Z-Wave network to avoid conflict between real nodes and decoys. Simulated traffic is generated thanks to loaded records of legitimate communication and is cyclically transmitted. Attackers can see multiple devices after sniffing communication, and they cannot determine with certainty which are real and which are virtual. Attempts to communicate with virtual decoys are logged and are the method of anomaly detection, which has a low rate of false positive alarms. Communication with virtual decoys seems to be legit because of responses generated by the IoT honeypot. Responding using basic frames like acknowledgments or reports convinces attackers about a successful attack. However, their presence is revealed. Interac-

tive communication with virtual decoys can satisfy an attack and keep him out of real devices. Network information about real devices and decoys are saved.

- **Non-interactive mode**

  The non-interactive mode is very similar to the default mode, but without interaction with attackers. The simulated traffic is still generated, but virtual decoys do not answer to commands sent by attackers. The main reason for usage of this mode can be the performance.

- **Passive mode**

  The passive mode does not allow for transmission. This mode provides a detailed view of Z-Wave traffic. The main reason for the usage of this mode can be a performance, a lack of transmitting hardware or a need for manual information gathering. Records and network information can be replicable, and it is necessary to know identifiers of Z-Wave networks. This mode displays all attributes of received frames.

3.3 Virtual Decoys

The best way to create an illusion of legitimate Z-Wave network using virtual decoys is to prepare frames, which represent simulated traffic of decoys. Frames can be saved and loaded as needed into *pcap* files. The IoT honeypot provides a set of virtual decoys, but extending this set will be simple thanks to the record mode. Figure 3.3 illustrates real usage of virtual decoys. However, this will be a single huge Z-Wave network from an attacker's point of view.

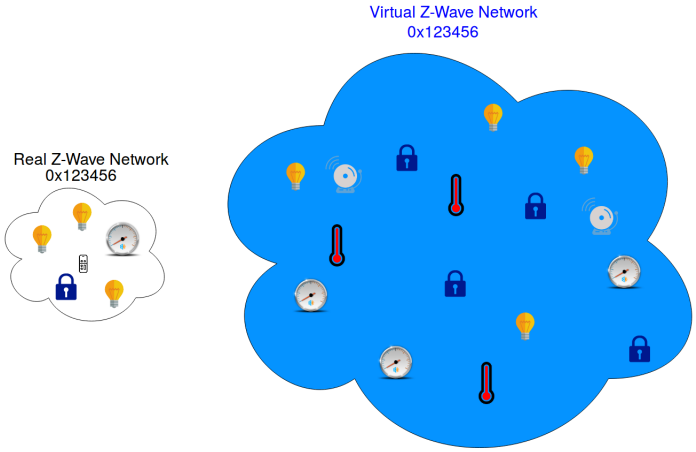

Figure 3.3: Real and Virtual Z-Wave networks

Network information about real devices and virtual decoys are saved to ensure persistent information, which can be used several times. The IoT honeypot offers to display this information. The IoT honeypot allows replication of virtual decoys using the special function. Existing networks can be extended and replicated which leads to creating a new network of decoys. It is recommended to map real nodes of a network, to which the user wants to add decoys to avoid a conflict.

3.4 Commands

This section discusses the main functions of the IoT honeypot that are available for a user as subcommands. All subcommands are accessible using one central command of the IoT honeypot. This command structure also represents a hierarchy of commands and their options:

```
command [options] subcommand [options] arguments
```

where every subcommand represents and implements different functionality of the IoT honeypot. Subcommands are listed here:

- ***record***

  This subcommand starts recording mode of the IoT honeypot. It receives and stores Z-Wave frames in internal data structures until the user exit. All nodes are virtualized in recorded frames and frames are saved into a *pcap* file before termination of the program. Network information and frames are collected into files, which path is set in the configuration of currently running instance of the IoT honeypot.

- ***run*** *[passive-mode] [non-interaction mode] [home-id]*

  This subcommand starts the main functionality of the IoT honeypot. It offers several options that can be used to specify a mode. Beside modes, this subcommand allows specifying a network identifier *Home ID*. The IoT honeypot tries to load recorded frames for this network. The IoT honeypot starts working with the network of first received frame, in case of leaving optional argument *home-id* blank.

- ***status***

  A subcommand ***status*** display saved information about networks that the IoT honeypot currently have. It shows network identifiers, identifiers of real nodes, virtual decoys and names of their records.

- ***read*** *file*

  This subcommand can be used for displaying recorded frames. Frames are recorded and saved into *pcap* files and their location is specified in

the configuration. An argument *file* is a target *pcap* file, which a user wants to display.

- ***replicate*** *home-id-A [home-id-B]*

  A subcommand *replicate* can be used for extending a network of decoys within the network *home-id-A*. Replication process consists of copying existing decoys and setting them new free node identifiers and double the network size. A network of decoys from *home-id-A* is copied to network *home-id-B* in case of setting the optional argument *home-id-B*. It is recommended to map real devices of network **home-id-B** before a process of replication begins. Two phases of the replication process are shown in Fig. 3.4

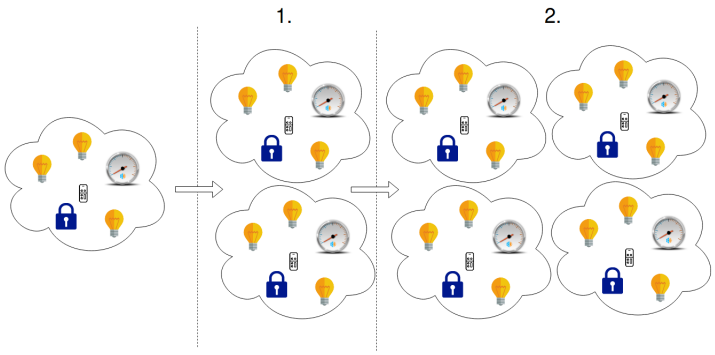

Figure 3.4: Two phases of an internal replication process of virtual decoys

- ***reset***

  This subcommand deletes all saved information about networks and all records. It is recommended to use it only in special cases. The IoT honeypot can handle the situation when a new real device is added to a network, and its node identifier is taken by a decoy. It erases the decoy and its records.

3.5 Data structures

This section describes the design of main data structures which are used in the IoT honeypot.

- **Configuration**

  The way of setting configuration is described in Sec. 3.1. A data structure of configuration includes information for multiple components. For example, the path where are records saved must be the same as the path, which is used for loading records. All main components of the IoT

honeypot need to be configured before the IoT honeypot starts working. Configuration class is illustrated in 3.5. The configuration object is initialized as passed to the monitor, the receiver, the traffic generator, the transmitter, and the controller sets its communication parameters – a frequency, a sample rate, and a TX gain.

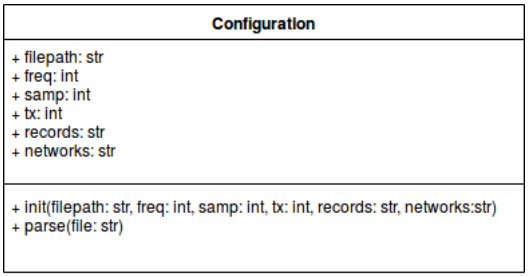

Figure 3.5: Configuration class

- **Networks**

  The IoT honeypot is divided into two processes - receiving and transmitting processes. It is necessary to share some information between these processes. The configuration is initialized before the start of the receiving and transmitting and shared to these processes without further changes during runtime. However, information about networks needs to be changed in some cases. One of these cases is adding a new real device to a network, which identifier overlaps with an identifier of an existing decoy. The existing decoy needs to be removed from the network data structure in both processes, and therefore it is necessary to apply this change in a network data structure of real devices and virtual decoys too.

  Not only network information needs to be shared between processes. Receiving *Home-ID* and passing it to the traffic generator in case of unspecified network identifier for running a standard functionality of the IoT honeypot described in Sec. 3.4. The receiver needs to be also informed, which frames were sent by the transmitter and which were not. Attackers can sniff and transmit the same frames as the honeypot generates and therefore the transmitter computes values of frames using a hash function. The transmitter passes all hash values of sent frames to the receiver. The same function computes hash values of received frames and recognizes self-sent frames. Sharing information between two main processes is illustrated in Fig. 3.6.

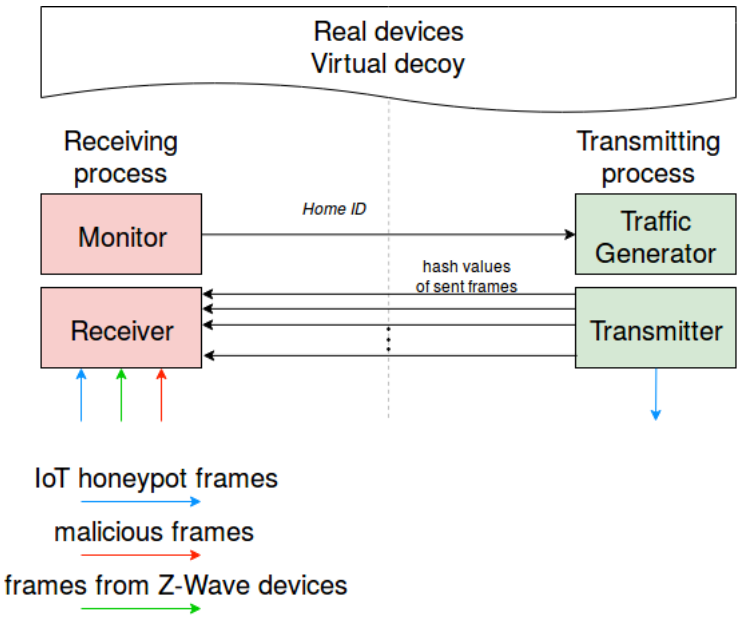

Figure 3.6: Processes and shared information

3.6 Detection method

A detection mechanism of malicious activities of the IoT honeypot is very simple and extends currently existing detection methods for unencrypted Z-Wave networks. Virtual decoys never belong to the production area. Thus any external manipulation with them can be considered as an attack attempt. The receiver is designed to recognize three main types of Z-Wave frames:

1. IoT honeypot frames,
2. frames from Z-Wave devices,
3. malicious frames.

The receiving process includes several decision points, which are shown in Fig. 3.7. First, *Home ID* is compared to *Home ID* of current configuration. The IoT honeypot works with frames of a single network. Next step leads to computing CRC. The frame is logged as invalid and is a subject to further analysis. The invalid frame is counted to special statistical counters if its destination is one of decoys and information like *Home ID*, node identifiers and command seems valid. Several devices don't drop invalid frames, so even the invalid frame can be potentially dangerous. Destination node identifier of the frame is compared with identifiers in a list of virtual decoys if the frame is valid. The real source

of the frame is either the IoT honeypot or an attacker in case of finding node identifier in the list of decoys. Finding duplicate of a received frame which has also been sent by the IoT honeypot leads to a detection of an attack, logging it and responding to the attacker. The IoT honeypot stores last ten sent frames in a queue. The frame is not considered malicious if it was received once and its source is the IoT honeypot. An attack is detected and logged if the frame destination is one of the decoys and has not been sent by the honeypot. Malicious frames are analyzed by *Monitor*, according to their intentions. Basic malicious frames can be divided into three main groups - frames that change a state of a device, frames that scan the state of the device and frames that fake the state of a device. *Monitor* recognizes the intentions of malicious frames and increments its counters. Receiving process can be stopped by a user.

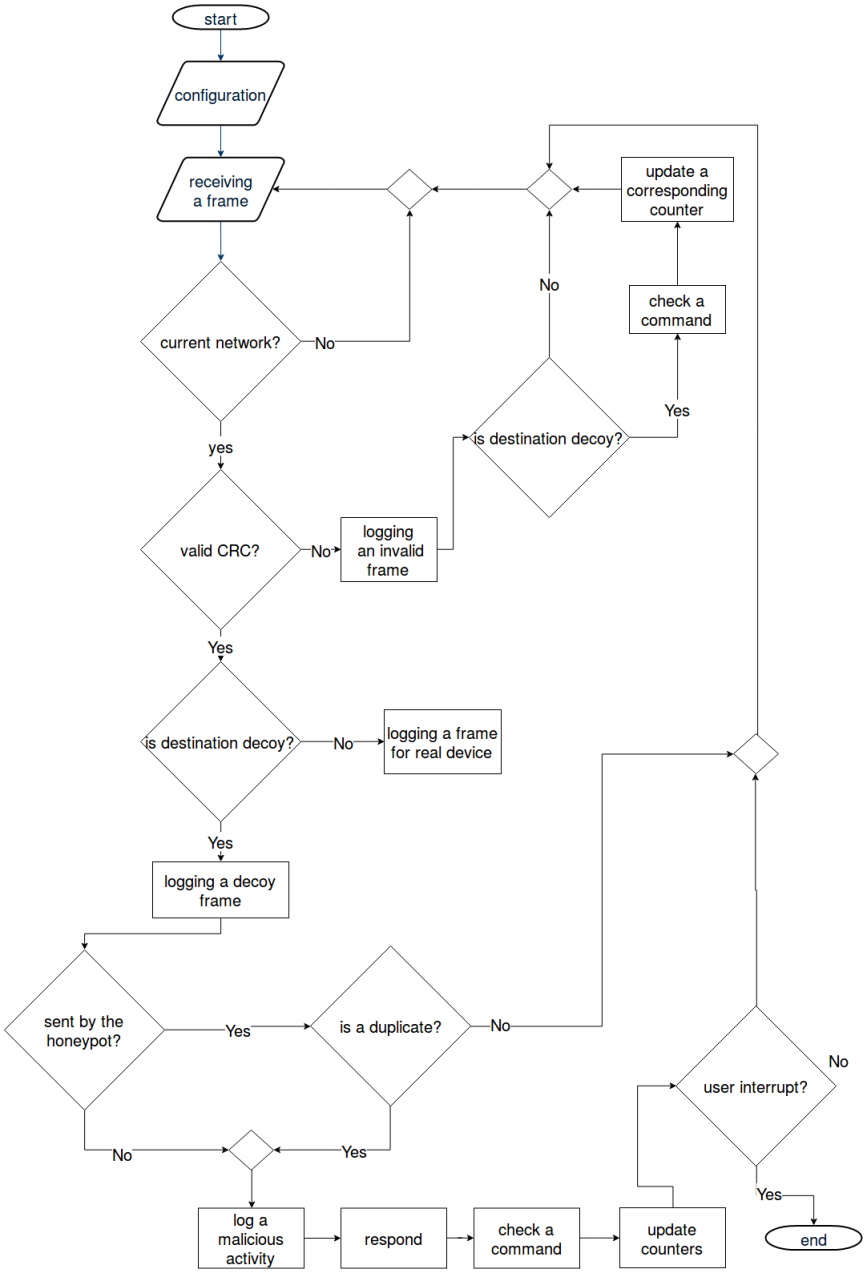

Figure 3.7: Attack detection of the IoT honeypot

Chapter 4

Realization

This chapter describes the realization of several main components and mechanisms. Python was used for the implementation of the IoT honeypot because it offers frameworks that are essential for this work. The IoT honeypot is a Python package that consists of code which uses several other packages and frameworks. The architecture of the system is described in Sec. 3.1. The final implementation uses RTL-SDR dongle for receiving and HackRF one for transmitting. Scapy-radio allows effective manipulation with the hardware components and frames.

Mainly used functions of Scapy-radio are the function for receiving and function for transmitting frames. Function for receiving supports multiple arguments like *timeout* and *store*, but for receiving Z-Wave frames need to be just specified protocol and handler function:

```
sniffradio(radio='Zwave', prn=lambda p: handle(p))
```

This function calling is an example, how receiving can start. Function *handle* takes as an argument *p*, which is received frame.

4.1 Control

This section explains the IoT honeypot control system, its implementation, fundamentals, and used packages. The file *Contoller.py* unites all commands that are implemented for manipulation and usage of functions. The IoT honeypot uses command line interface (CLI) which was easily implemented thanks to a Python package Click [80]. Click is highly configurable but comes with sensible default out of the box. It offers parsing options and arguments, arbitrary nesting of commands and automatic help generation. The main command and subcommands are implemented according to Sec. 3.4. Thus there is one main command *iotpot* with options and a set of subcommands:

```
iotpot [options] subcommand [options] arguments
```

Options of *iotpot* are essential and inherited for all subcommands:

- *–config/-c* - a path to a configuration file
- *–freq/-f* - a frequency of a receiver and a transmitter
- *–samp/-s* - a sample rate of the receiver
- *–tx/-t* - a transmission gain (TX) of the transmitter
- *–record/-r* - a path to records
- *–networks/-n* - a path to persistent network information
- *–log/-l* - a path to a logging file
- *–alerts/-a* - a path to a alerts file

Some options have its default values and more detailed information can be found in the file *CONSTANTS.py*. Values are loaded from a configuration file if options are left blank. Main command *iotpot* initialize a logger and configuration. Click allows passing context to subcommands using the standard Python *dict* structure. The object of *Configuration* and *logger* are passed to context of subcommands:

```
ctx.obj[LOGGER] = iotpot_logger
ctx.obj[CONFIGURATION] = configuration
...
logger = ctx.obj[LOGGER]
configuration = ctx.obj[CONFIGURATION]
```

where *ctx* is the context consisting of *dict* data type shared between the main command and all subcommands.

4.1.1 Configuration

Configuration of the IoT honeypot can be executed by options of *iotpot* command as described in Sec. 4.1. However, a more suitable way of configuration is using a configuration file, in which a user can save settings of the IoT honeypot. The Python module Configparser [81] allows parsing of configuration files that use a human-readable structure for data representation. The IoT honeypot defines a few sections and keys that can be used in a configuration file. The example of a configuration:

```
[communication]
frequency = 868420000
sample_rate = 2000000
tx_gain = 25

[recording]
records_path = /path/to/records

[networks]
network_path = /path/to/networks

[logging]
logging_path = /path/to
alerts_path = /path/to
```

Configuration of Scapy-radio, so the receiver RTL-SDR dongle and the transmitter HackRF One is a separate process. First, Scapy-radio must load GNU Radio using command *load_module('gnuradio')*. Applying settings is possible when a module is loaded, and loading takes a few seconds. The configuration process is trying to set the configuration until it is not successful.

Setting communication parameters of Scapy-radio and hardware is available thanks to commands of Scapy-radio. These commands can be executed during runtime. GNU Radio Companion allows the definition of custom variables that are accessible using their names. GNURadio Companion blocks of the IoT honeypot has multiple variables defined – central frequency, sample rate, and TX gain. Variables can be defined by blocks, as shown in Fig. 4.1.

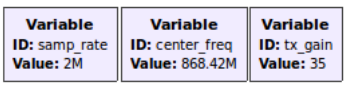

Figure 4.1: Blocks of variables

These variables are set from a Python code using commands:

```
gnuradio_set_vars(center_freq = configuration.freq)
gnuradio_set_vars(samp_rate = configuration.samp_rate)
gnuradio_set_vars(tx_gain = configuration.tx)
```

All communication parameters are displayed immediately after a successful configuration. The success of configuration is checked thanks to commands that read variables from GRC blocks after setting them. One of this command is for example:

```
gnuradio_get_vars('center_freq')
```

The option of changing the receiving frequency offset and transmitting frequency offset remains, as GRC allows usage of sliders. Sliders are elements of a graphical user interface, that is automatically generated. Sliders offer effective control over variables. Addition of sliders is also modification of original EZ-Wave GRC blocks for Z-Wave.

4.1.2 Subcommands

Subcommands represents the interface for functionalities of the IoT honeypot. A hierarchy of the main command, options, subcommands, and arguments are described in Sec. 3.4.

4.1.2.1 record

This subcommand loads network information and virtual decoys from *json* data-structures into standard Python *dict* data types. Configuration and a logger are loaded from a shared context *ctx*. Objects of *Receiver* and *Monitor* are initialized with *network*, *decoys*, *logger* and *configuration*. Recording starts function *self.receiver.start(recording=True)* called by *Monitor*. The Scapy-radio function *sniffradio* use as an argument a frame handler function of the*Receiver* object that is specialized for recording of frames. Every received frame is validated using CRC. Since legitimate communication of real Z-Wave devices is recorded, existing decoys with possible duplicates of node identifiers are removed within their records to avoid an identifier conflict among nodes. Unique identifiers of Z-Wave devices are saved into *networks*, where are divided into groups according to *Home ID*.

All recorded frames are displayed and saved into a temporary data structure. All recorded frames are modified when receiving stops. The IoT honeypot computes set of unused node identifiers from existing Z-Wave devices and decoys in a current network. Every unique *Node ID* in frames is temporary mapped and swapped for a different free node identifier to create virtual decoys. Modified frames are saved using command *wrpcap* to a directory specified in *Configuration*. Information about virtual decoys is saved into persistent data structure *decoys*, where decoys are divided into groups according to *Home ID*. Every virtual decoy is represented by a state, a list of occurrences in records and its *Node ID*. It is recommended to check which Z-Wave devices were recorded to avoid malicious activities. Unlike design, the implementation allows only two cases of network discovering and building a network-information data structure. The recording is one of them. Steps of the recording process of legitimate communication that leads to the creation of decoys are illustrated in Fig. 4.2.

Every new recording on the same network creates new decoys because decoys are not firmly mapped to real devices. The main reason is to achieve

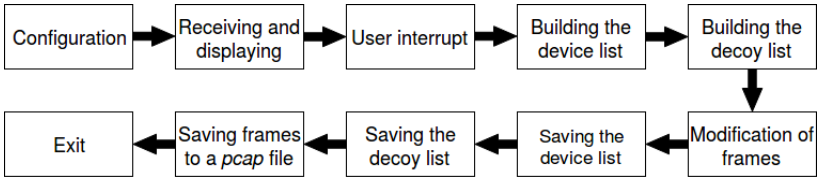

Figure 4.2: Recording communication and creating decoys

more variability, complexity, and scalability of generated traffic and to allow the creation of different use cases between seemingly different nodes.

4.1.2.2 run *[passive mode] [non-interative mode] home-id*

This subcommand being with loading the configuration and the logger from the shared context. Passive mode offers to receive and displaying Z-Wave frames into a terminal.

Default mode uses *Manager* from the package *multiprocessing* [82]. *Manager* handles sharing standard Python *dict* structures containing network and decoys information between two main processes - receiving and transmitting. *Receiver*, *Monitor*, *Transmitter*, *TrafficGenerator* and *Responder* - one object of each is created and initialized. Two instances, which are illustrated in Fig. 4.3 of *Pipe* from *multiprocessing* are created and passed to other components. One pipe is for sharing *Home ID* from *Monitor* to *TrafficGenerator*. *Monitor* reads this *Home ID* from first received frame in case of empty argument and send it to *TrafficGenerator*. *TrafficGenerator* use this information to load corresponding records. Another pipe is used for sharing hash values of sent frames from *Transmitter* to *Receiver*, as illustrated in Fig. 4.3. Pipes are suitable for this usage, as they offer more faster processing than *Manager*.

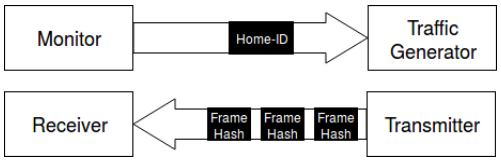

Figure 4.3: Two Pipes used for communication

There start two new processes during running in a default mode. One separated process is for receiving, and another one is for setting configuration. The main process ensures transmitting. *Monitor* starts receiving function of *Receiver*. Receiver check CRC and *Home ID*. *Home ID* is sent to *TrafficGenerator* by *Monitor* in case of non-specified network identifier as an argument. A hash value of a received frame is calculated if its destination is one of the virtual decoys. *Receiver* keeps hash values of last ten sent frames in the queue *decoy_frames_out* and all hash values of received frames in *decoy_frames_in*.

Monitor detects and log attack, as illustrated in Fig. 4.4, because the received frame is sent to a decoy and was not sent by *Transmitter*. *Monitor* logs all received frames.

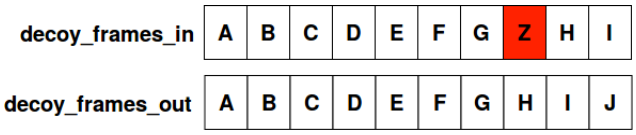

Figure 4.4: Malicious modified frame detected

A replay attack is detected and logged if a duplicate of a hash value of the received frame was already found in *decoy_frames_in*. This queue remembers the last ten frames sent by the IoT honeypot, which means two same frames for a decoy was received, as illustrated in Fig. 4.5. It is certain that one of them was sent by the attacker because the IoT honeypot never sends the same frames in such a short time interval. The frame is appended to the queue *decoy_frames_in* if it was received for the first time.

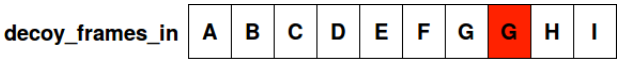

Figure 4.5: Malicious duplicate of one of last ten frames sent by the honeypot

Responder prepares answers for attackers if an attack attempt is detected. Responds consist of acknowledgments frames and reporting frames that are usually used in legitimate communication. These frames are sent by *Transmitter*. Non-interactive mode of the IoT honeypot disable functionalities of *Responder*.

4.1.2.3 replicate *home-id-A [home-id-B]*

The *replicate* subcommand provides replication of records and decoys, as designed in Sec. 3.4. The first argument refers to a source network identifier – its corresponding decoys and records that will be replicated. Another argument is optional and refers to a destination network identifier - to which network data will be replicated. Leaving the second argument blank leads to replication of network *home-id-A* internally. First, configuration, network, and decoys are loaded. A set of free nodes of a target network is computed as a complementary set of a unified set of all existing decoys and devices in the target network. All decoy identifiers from the source network are mapped to new numbers. Records are modified, and all information is saved after the replication process. It is recommended to do replication only when real devices are already recognized by the IoT honeypot. For example, Network B lacks variability and count of devices. Decoys from more complex Network A can be

replicated to Network B, as illustrated in Fig. 4.6. Replication can be internal or external. Controllers are replicated to, to make generated traffic more complex and variable, as Z-Wave networks can include several controllers.

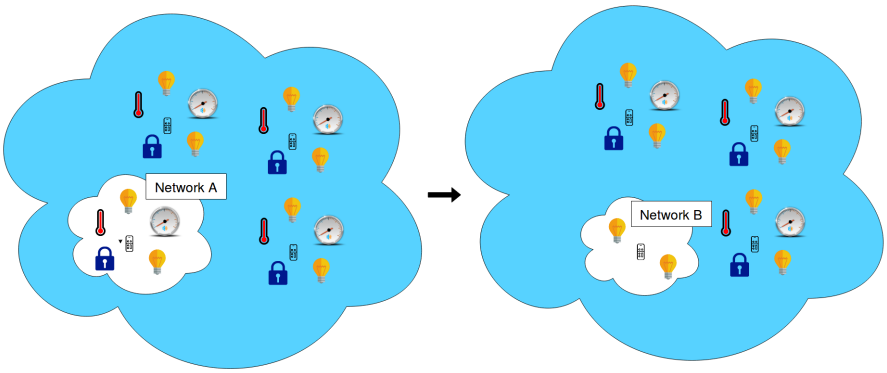

Figure 4.6: External replication of decoys from Network A to Network B

4.1.2.4 Other subcommands

The implementation of the IoT honeypot offers more subcommands, and some of them differ from a design.

- **status**

  This subcommand display status of the persistent data of the IoT honeypot. It divides identifiers of devices and decoys into groups by networks.

- **reset**

  This subcommand removes all persistent information and records of the IoT honeypot. It is recommended to use it only in special cases.

- **add *home-id***

  This subcommand is an addition to a design. This subcommand allows adding a single new real device to a network. Adding a single device and recording are the only two situations when network devices can be discovered and added to the IoT honeypot. Allowing discovering new devices during a default mode could bring vulnerabilities. For example, all unused node identifiers for decoys could be used for fake devices, if an attacker transmits frames with various *Node ID* values. Adding a single device starts with receiving. The new device is added if a frame from unknown device is received. The argument *home-id* is required.

- **delete *home-id node-id*** This subcommand deletes virtual decoy that has identifier *node-id* in network *home-id*. All corresponding records are deleted too.

- **read *record*** This subcommand load and display recorded frames in *pcap* file selected as an argument *record*.

4.2 Logging

Logging is an essential activity of the IoT honeypot. Logging provides a representation of results in the terminal and special files. The Python module *logging* [83] defines functions and classes which implement a flexible event logging system for applications and libraries, as it is described in the official documentation.

The IoT honeypot uses a single logger, but several handlers. The first handler ensures logging to a terminal. Major output in the terminal is printed by the logger. The second handler saves all logging messages to a file *iotpot.log*. Third handler prints all alerts to a special file *alerts.iot*. Locations of these files can be specified using options of the main command *iotpot*. Basic statistics informing about several types of received frames are printed after user interrupt.

The logger allows dividing logging messages by levels. Standard levels are *DEBUG, INFO, WARNING, ERROR* and *CRITICAL*. Even there is a possibility of making custom level, the IoT honeypot uses the *DEBUG* level for logging mainly results of CRC computations and received frames representations. The level *INFO* has higher priority than *DEBUG*, and its task is to inform about more rare events like setting communication parameters. The level *WARNING* is used for logging alerts that warn about attack attempts. Time and date are added to all logged messages. First handler uses *ColoredFormatter* to highlight level names.

4.3 Persistence of data structures

The IoT honeypot needs to store some of the gathered information to accomplish reusability of data. Information that needs to be stored are records of legitimate communication, information about real Z-Wave networks and information about sets of decoys corresponding to individual networks.

Records of frames are saved into *pcap* files. Scapy-radio offers function *wrpcap* to save frames to files and *rdpcap* to read records from files to variables. Since records are saved into *pcap* files, installing special dissectors that are described in [64] can these frames be recognized by Wireshark.

Network information and information about decoys are saved into *json* files. Structure of network information is simple and consists of networks

with their node identifiers. The example of network information saved in *real_networks.json* file:

```
{
"0xdf11f630":
        [1, 8, 9, 10],

"0xaa08f623":
        [1, 25, 38, 42],

"0xcd07a510":
        [1, 38, 39]
}
```

Information about virtual decoys is more complex. The example of information about decoys saved in *virtual_networks.json* :

```
{
"0xdf11f630":
        {"2":
                {"records":
                        ["20190427-161336.pcap"],
                "state": "controller"},
        "174":
                {"records":
                        ["20190427-161336.pcap"],
                "state": "\u0000"}
        }
}
```

4.4 Interaction

The IoT honeypot provides interaction to attackers. The main reason for this interaction is to pretend that virtual decoys are legitimate Z-Wave devices and get more time to detect and react. The IoT honeypot can detect several types of malicious frames destined for its decoys. Detection leads to simple responding, which was implemented according to the analysis of legitimate communication between Z-Wave device and controller. The vast majority of ordinary communication consists of frames with commands *GET*, *SET*, frames including *REPORT* of device's state and *ACK frames*. Malicious activities like taking control over a device or scanning its state are done by frames that include *SET* or *GET* commands. *Responder* is able to respond to attackers using *ACK* and *REPORT* frames, as shown in Fig. 4.7. Beside responding, state of decoys changes if attackers try to manipulate it.

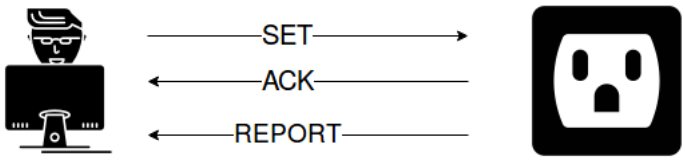

Figure 4.7: Interaction between an attacker and a Z-Wave outlet

4.5 Validation

There are several situations when the IoT honeypot needs to validate a frame. Z-Wave uses CRC-16 for validation of frames, and therefore CRC is computed from all received frames. Value of CRC is included in every frame, and the IoT honeypot calculates its value of CRC and compares them. Same values of CRC means the received frame is valid. Preparing a frame to be sent is another situation when computing CRC is necessary. All transmitted frames need to have correct CRC to seem legitimate enough. Operation XOR is used in a function that ensures the calculation of CRC.

Another kind of validation that is used is a hash function. Hash values of frames are passed from transmitter to receiver to filter frames sent by the IoT honeypot. The package *xxhash* and its function *xxh32* are used for computing hash values of frames.

4.6 Data Set of Communication

This thesis provides network information, several records, and virtual decoys. This data set should serve if a user doesn't have Z-Wave devices and thus it is unable to record communication. This data set makes the IoT honeypot usable even without a real Z-Wave network. This data set consists of legitimate communication between three Z-Wave nodes, as shown in Fig. 4.8. However, implementation of recording mode and replication process allows creating complex legitimate generated traffic, as every successful recording creates a new set of decoys and use-case.

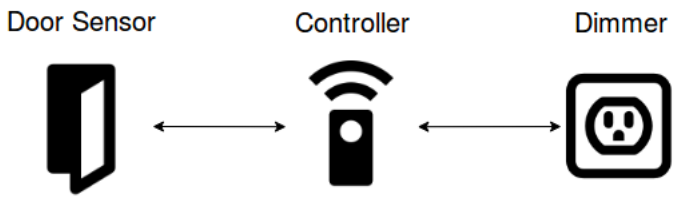

Figure 4.8: Real network consisting of a door sensor, a controller and a dimmer

Data set includes network information, several records in *pcap* files and

virtual decoys. Records represent multiple use-cases in real Z-Wave network. The identifier *Home ID* of this network is *0xdf11f630*.

Chapter 5

Testing

This chapter summarizes the testing of a prototype of the IoT honeypot. First, a test environment is described. Testing is focused on demonstrating the functionality of the IoT honeypot, its qualities and reveal its defects.

5.1 Test Environment

The IoT honeypot uses RTL-SDR dongle for receiving and HackRF One for transmitting. Both devices use an antenna. All testing is done using a single transmitter. The prototype of the IoT honeypot is the Python package that is installed on a notebook. The notebook has four CPU cores, 6 GB of RAM memory and the 64-bit operating system Ubuntu 16.04 LTS. The IoT honeypot has records of decoys that are prepared to operate in a real Z-Wave network. Testing a Z-Wave network consists of a controller, a dimmer and a door sensor. Records of decoys have been recorded using a recording mode of the IoT honeypot. Two types of deployment are possible. The IoT honeypot can be deployed within an existing Z-Wave network or without the Z-Wave network. The analysis presents and describes two tools that can be used for penetration testing - Scapy-radio and waving-z in Sec. 2.2.1. We have only one transmitter available. A special option was added to the IoT honeypot to allow using Scapy-radio for the functionality of the IoT honeypot and testing at the same time. Beside penetration tests, several functional tests are executed too.

5.2 Testing Scenarios

This section proposes several testing scenarios of the IoT honeypot.

5.2.1 Detection Method Testing

Tools discussed in analytic part of this work in Sec. 2.2.1 are sufficient for testing of the detection method. The IoT honeypot is a security element, and it should be able to detect malicious frames. Testing of the detection method should include real transmission of malicious frames. Scapy-radio, waving-z, and hackrf_transfer will be used for transmission, modulation, and encoding of malicious frames. A homogeneous set of malicious frames will be used as the first type of tests. Another set should contain common frames and a certain percentage of malicious frames, to prove detection during the normal mode of the IoT honeypot. The detection method will be tested during the non-interactive mode of the IoT honeypot and during the interactive mode too. Since there are two types of deployment, tests should provide results using the honeypot within the Z-Wave network and without that. Testing of the detection method will be sophisticated. It will provide information about the quality of reception, transmission, the detection method, logging, statistic counters and ability of coexistence with the network.

5.2.2 Passive Mode Testing

The passive mode of the IoT honeypot is a simple receiving of frames and displaying them in human-readable form. This mode provides informative functionality for a user. Transmitting several sets of frames should lead to displaying received frames, dividing them into groups according to type and increasing corresponding statistical counters. Beside receiving of frames from real Z-Wave devices, reception of frames sent by SDR will be tested too.

5.2.3 Responding Mode Testing

Since the IoT honeypot can interact with attackers to some extent. Basic frames are sent as responds to create a simple illusion of legitimate devices. Testing of responding will be done by transmitting malicious frames and counting resnponses.

5.2.4 Replication Testing

The IoT honeypot offers the function of replication, that can effectively replicate existing records and information originated in one network to another. Replication can be used in case of missing variability and complexity of the Z-Wave network, thus the inability of the recording of high-quality records and information. Replication can also be used for usage of the IoT honeypot without the Z-Wave network. Replication will be tested using its functionality and checking new replicated data.

Table 5.1: Detection of 100 malicious frames prepared by waving-z

Number of Frames	**SET**	**GET**	**REPORT**
all sent frames	100	100	100
received valid malicious	90	86	76
recognized valid frames	90	86	76
invalid	0	0	0

5.2.5 Other Functionalities

Since the honeypot provides informative functions like *read* or *status* and record-management functions like *add* or *delete*, these functionalities will be tested too. Simple usage of these functions ale checking an output should provide enough information to measure their quality, because implementation of these function is elementary.

5.3 Test results

Several tests were performed according to the proposed testing scenarios in Sec. 5.2.

5.3.1 Detection Method testing

There are two sets of tools that can be used for transmitting. Encoding and modulating frames using waving-z and transferring them using the program hackrf_transfer ensures results of the first test. There were created three sets of malicious frames with different commands. The first set represents frames, that can manipulate a state of devices - frames with the *SET* command. Frames with the *GET* command can be used for device's state scanning. Frames with the *REPORT* command can be used for faking device's state. Every set consists of one hundred malicious frames. The IoT honeypot receives these frames during non-interaction mode without traffic-generation. These special conditions were ensured because of the lack of a second transmitter. Table 5.1 shows the results of detection of transmitted 100 malicious frames with each command. The table informs about several successfully received frames and the number of frames whose commands have been recognized.

The second test works with the same data sets and honeypot settings which were used in the first test. However, instead of waving-z and hackrf_transfer, these frames are sent by scapy-radio. Since the IoT honeypot recognizes potentially malicious invalid frames, the Tab. 5.2 includes them too. This table provides apparent differences in the reliability of transmission between hack_rf and scapy-radio. Recognition of frames is successful.

Table 5.2: Detection of 100 malicious frames transmitted by scapy-radio

Number of Frames	SET	GET	REPORT
all sent frames	100	100	100
received valid malicious	49	55	48
recognized valid frames	49	55	48
invalid and potentially malicious	4	4	1
all invalid frames	12	13	15

Table 5.3: Detection of 100 malicious frames transmitted by scapy-radio

Number of Frames	SET	GET	REPORT
all sent frames	100	100	100
received valid malicious	58	61	67
recognized valid frames	57	61	67
all valid decoy frames	114	121	83
invalid and potentially malicious	3	4	2
all invalid frames	25	22	13
unrecognized	1	0	0

The third test works with the same data sets as the second test. The only difference is the interaction of the IoT honeypot. Since the IoT honeypot did not interact in the second test, it responds to malicious frames in the third test. Tab. 5.3 presents information about received and recognized frames during this test. Beside categories from the second test, the third test shows a summary of all recorded decoy frames in which are responds counted. One unrecognized frame appears during this test too.

The fourth test uses sets of 100 frames. However, only every fifth is malicious. Thus every set includes 20 malicious frames. Results of this test are summarized in Tab. 5.4. The IoT honeypot was responding to malicious frames.

The fifth test uses sets of 100 frames, and every fifth frame is malicious. This test was executed in the presence of real Z-Wave network to test coexistence of the IoT honeypot within the real network. Tab. 5.5 summarizes results. Usage of the real Z-Wave network was random. The IoT honeypot counted frames from real devices too.

5.3.2 Passive Mode

Data sets of malicious frames are used for passive mode too. Passive mode ensures a more detailed representation of received frames to satisfy require-

Table 5.4: Detection of 20 malicious frames transmitted by scapy-radio with interaction

Number of Frames	**SET**	**GET**	**REPORT**
all sent frames	100	100	100
sent malicious frames	20	20	20
received valid malicious	7	13	8
recognized valid frames	7	13	8
all valid decoy frames	65	71	61
invalid and potentially malicious	5	4	4
all invalid frames	21	14	8

Table 5.5: Detection of 20 malicious frames transmitted by scapy-radio with interaction and real Z-Wave network

Number of Frames	**SET**	**GET**	**REPORT**
all sent decoy frames	100	100	100
frames from real devices	42	63	14
sent malicious frames	20	20	20
received valid malicious	8	16	8
recognized valid frames	8	16	8
all valid received decoy frames	64	71	57
invalid and potentially malicious	3	1	6
all invalid frames	19	16	12

ments of a potential deeper analysis of frames. All received decoy frames were sent by program hackrf_transfer. Test results of passive mode are shown in Tab. 5.6.

5.3.3 Responding

This test use data sets of 100 malicious frames. The main goal of the test is to count responding frames. The IoT honeypot responds *ACK* and *REPORT* frames. Tab. 5.7 summarizes results. This table includes numbers of responses. Zero *REPORT* responds to *REPORT* frames are logical.

Table 5.6: Detection of 20 malicious frames transmitted by scapy-radio with interaction and real Z-Wave network

Number of Frames	SET	GET	REPORT
all sent decoy frames	100	100	100
frames from real devices	48	26	51
sent malicious frames	100	100	100
received valid malicious	55	81	85
recognized valid frames	55	81	885
all invalid frames	4	0	0

Table 5.7: Responding to 100 malicious frames transmitted by scapy-radio with interaction and real Z-Wave network

Number of Frames	SET	GET	REPORT
frames from real devices	5	13	21
sent malicious frames	20	20	20
received valid malicious	9	8	9
recognized valid frames	9	8	9
all invalid frames	18	21	8
invalid and potentially malicious	7	3	7
ACK responded	25	24	17
REPORT responded	15	11	0

5.4 Replication

The main goal of this test was to check if replication ensures the creation of a new set of decoys and records for a network specified as an argument. The source network is *0xdf11f630* and target network *0xdf11f640*. Even when the IoT honeypot misses information of real devices network of the *0xdf11f640* network, the replication process should create a usable set for the main functionality of the IoT honeypot.

Subcommand *status* provided information about a single network before the replication process: The replication process created sets for the new network and the *status* subcommand shows information about the new network and its decoys.

5.5 Summary

Functionality and subcommands like *status, add, delete* and *read* were tested during complex testing of other functionalities. The subcommand *status* is able to provide information about persistent information gathered or created by the honeypot. Subcommands *and* and *delete* are usable for node management. Subcommand *read* was tested and is able to display recorded frames in human readable form.

Testing of detection method, passive mode and responding provided useful information that can be used for future work. Tests showed the program *hackrf_transfer* has higher reliability of transmitting than Scapy-radio. However, the IoT honeypot uses Scapy-radio because of its number of functions like frame representation, frame recognition, recording into *pcap* files, the ability of transmission and reception at once, the flexibility of its GRC blocks, configurability during run time and many more. Lower reliability of the transmission of Scapy-radio needs further analysis of its lower layers. On the other hand, even when *hackrf_transfer* offers more reliable transmitting, frames need to be first modulated and encoded to individual files before transmission. The lack of flexibility, configurability during run time, the inability of frame recognition makes Scapy-radio more realistic choice for the IoT honeypot.

Conclusion

One of the goals of this thesis was to analyze the state-of-the-art of hardware and software tools for receiving, processing and transmitting signals of wireless protocols for the Internet of Things. This thesis introduced and summarized general information about the current state of the Internet of Things and currently available hardware and software.

The analytic part of this thesis is focused on the analysis of the protocol Z-Wave that is used for home automation and its security vulnerabilities. Beside generation information about Software Defined Radio, a flexible solution for digital signal processing, this part explains its usage in the context of penetration testing of wireless protocol Z-Wave, basic principles of its operating and several hardware and software solutions are introduced. Several ways of basic attack that is based on analyzed vulnerabilities of Z-Wave are summarized too. The last part of the analysis provides general information about honeypots and existing solutions of IoT honeypots, as potential security elements of IoT networks.

The design of the IoT honeypot discusses a system's architecture and its components. The design of a prototype is based on information gathered from analysis and considers Software Defined Radio hardware as essential components. The IoT honeypot is a device that can receive and transmit Z-Wave frames. A role of a receiver takes up the RTL-SDR dongle. HackRF One is designed to be used as a transmitter. Other components of the IoT honeypot are designed to monitor malicious activities, store reusable and valuable information and interact with attackers to some extent. The main idea of the principles of the IoT honeypot is to create a set of targetable and authentic decoys that will lure potential attackers. Last part of this chapter includes test scenarios.

Since the design of the IoT honeypot was done properly, there were only a few problems during implementation. Realization explains the usage and integration of several hardware and specific software components and packages. Implementation of main functionality like detection method, interaction,

logging and recording are summarized too. Creation of communication data sets is one of the essential goals of this thesis. The IoT honeypot receives legitimate communication of real Z-Wave devices and produces records and network information about virtual decoys, which are reusable and replicable.

Tests have provided information about the quality of certain functions, system shortcomings and deployability within the real Z-Wave network. Complex tests prove that main functions of the IoT honeypot work satisfactorily, but are hampered by low reliability of the used tools. The lack of hardware components and the complexity of used tools disallow us to detect reason unreliable transmission. The extremely low false-positive rate of detection method is achieved thanks to the main idea of the IoT honeypot. However, the IoT honeypot is not intended as a single security element of Z-Wave networks. Its main contribution is to increase changes of attack detection, prevention against malicious activities and helps to patch vulnerabilities of existing Z-Wave networks with unencrypted communication.

The output of this thesis is a working prototype of the proposed IoT honeypot that is deployable on real Z-Wave networks. The prototype was tested on a set of real Z-Wave devices. Hardware tools for this thesis were available thanks to cooperation with the organization CESNET.

Future Work

This thesis provides analysis of an available set of tools for digital signal processing, Software Defined Radio, which is very flexible and therefore can be used in a wide range of applications. However, its flexibility reduces the ease of use, because deeper knowledge is needed to create a custom solution. Implementing GRC blocks for all sets of communication parameters of Z-Wave could increase the quality of reception and transmission. Further optimization and more reliable hardware components could bring better results and more reliable transmission. Another improvement could include the implementation of more advanced detection methods supporting a more extensive range of command classes of frames. Statistical methods and machine-learning could allow more advanced methods of traffic generation, detection and responding.

Bibliography

[1] Ashton, K. That 'Internet of Things' Thing. *RFiD Journal*, volume 22, January 2009: pp. 97–114.

[2] State of the IoT 2018: Number of IoT devices now at 7B - Market accelerating. *IoT Analytics GmbH [online]*, August 2018, [cit. 2018-08-2018]. Available from: `https://iot-analytics.com/state-of-the-iot-update-q1-q2-2018-number-of-iot-devices-now-7b`

[3] Schmid, T.; Dreier, T.; et al. Software radio implementation of short-range wireless standards for sensor networking. January 2006: pp. 119–120, doi:10.1145/1182807.1182865.

[4] *A comprehensive methodology for deploying IoT honeypots*, *Lecture Notes in Computer Science*, volume LNCS 11033, Regensburg (Germany): Springer Nature Switzerland AG, September 2018, doi:10.1007/978-3-319-98385-1_16.

[5] Swamy, S. N.; Jadhav, D.; et al. Security threats in the application layer in IOT applications. In *2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, February 2017, pp. 477–480, doi:10.1109/I-SMAC.2017.8058395.

[6] Rehman, A.-u.; Mehmood, K.; et al. Communication Technology That Suits IoT - A Critical Review. April 2013, doi:10.1007/978-3-642-41054-3_2.

[7] ZigBee Alliance. *Zigbee Specification*. September 2012.

[8] Jia, N.; Zheng, C. Design of Intelligent Medical Interactive System Based on Internet of Things and Cloud Platform. In *2018 10th International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC)*, volume 01, August 2018, pp. 28–31, doi:10.1109/IHMSC.2018.00015.

[9] Sornin, N.; Yegin, A. *LoRaWAN^{TM}1.1 Specification*. LoRa Alliance, Inc., 3855 SW 153rd Drive, Beaverton, OR 97003, October 2017.

[10] Silicon Labs. *Software Design Specification*. April 2019.

[11] Cagan, K.; Lisiak, J.; et al. LoRa@FIIT Specification. STU FIIT, April 2017.

[12] Sigfox, Bâtiment E-volution425, rue Jean Rostand31670 Labrège – France. *Sigfox Technical Overview*. April 2017.

[13] Wedd, M. What Is Cellular IoT? August 2018. Available from: `https://www.iotforall.com/what-is-cellular-iot/`

[14] Rama, Y.; Özpmar, M. A. A Comparison of Long-Range Licensed and Unlicensed LPWAN Technologies According to Their Geolocation Services and Commercial Opportunities. In *2018 18th Mediterranean Microwave Symposium (MMS)*, October 2018, ISSN 2157-9830, pp. 398–403, doi:10.1109/MMS.2018.8612009.

[15] Mohamed Abomhara, G. K. Cyber Security and the Internet of Things: Vulnerabilities, Threats, Intruders and Attacks. *Journal of Cyber Security*, volume 4, May 2015: pp. 65–88, doi:10.13052/jcsm2245-1439.414.

[16] Adelantado, F.; Vilajosana, X.; et al. Understanding the limits of LoRaWAN. *IEEE Communications Magazine*, volume 55, June 2017, doi:10.1109/MCOM.2017.1600613.

[17] Olsson, J. *6LoWPAN demystified*. Texas Instruments, October 2014.

[18] Sarawi, S.; Anbar, M.; et al. Internet of Things (IoT) Communication Protocols : Review. July 2017, doi:10.1109/ICITECH.2017.8079928.

[19] Texas Instruments. *RFID SystemsProduct Specifications*. 2010.

[20] Guillemin, P.; Berens, F.; et al. Internet of Things Standardisation - Status, Requirements, Initiatives and Organisations. January 2013, ISBN 9788792982735, pp. 259–276.

[21] Khalefa, M.; A Jabar, M.; et al. The Internet of Things Software Architectural Solutions. *Australian Journal of Basic and Applied Sciences*, volume 9, November 2015: pp. 271–277.

[22] Picod, J.-M.; Lebrun, A.; et al. Bringing Software Defined Radio to the penetration testing community. Black Hat, 2014.

[23] Mays, C. E. Constructing Honeypots to Defend Building Automation Systems. Technical report, AIR FORCE INSTITUTE OF TECHNOLOGY WRIGHT-PATTERSON AFB OH WRIGHT-PATTERSON AFB United States, March 2017.

[24] Minn Pa Pa, Y.; Suzuki, S.; et al. IoTPOT: A novel honeypot for revealing current IoT threats. *Journal of Information Processing*, volume 24, May 2016: pp. 522–533, doi:10.2197/ipsjjip.24.522.

[25] Telnet IoT honeypot. `https://github.com/Phype/telnet-iot-honeypot`, 2019, accessed: 2019-03-19.

[26] Arduino Uno Rev3. `https://store.arduino.cc/arduino-uno-rev3`, 2019, accessed: 2019-01-02.

[27] LoRa Shield for Arduino. `http://www.dragino.com/products/module/item/102-lora-shield.html`, 2019, accessed: 2019-01-02.

[28] Microchip, Microchip Technology Inc. 2355 W. Chandler Blvd, Chandler, Arizona, 85224-6199, USA. *LoRa® Technology GatewayUser's Guide*. 2016, user's guide for a LoRa gateway.

[29] LoRa Server, open-source LoRaWAN network-server. `https://www.loraserver.io/`, 2019, accessed: 2019-01-15.

[30] LoRa-FIIT. `https://github.com/HalfDeadPie/LoRa-FIIT`, 2017, accessed: 2019-01-15.

[31] Cagan, K.; Lisiak, J.; et al. Proposal of STIOT Protocol. STU FIIT, April 2017.

[32] PacketConverter. `https://github.com/mintos5/PacketConverter`, 2017, accessed: 2019-01-15.

[33] Evans, D. The Internet of Things - How the Next Evolution of the Internet Is Changing Everything. Technical report, Cisco Internet Business Solutions Group (IBSG), April 2011. Available from: `https://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf`

[34] Zensys. *Z-Wave^{TM}the wireless control language*.

[35] Silicon Labs. *Z-Wave Node Type Overview and Network Installation Guide*. March 2018.

[36] Z-Wave Alliance. *Z-Stick Gen5*. February 2017.

[37] POPP. *Z-Wave Wall Plug Type E Dimmer*. 2018.

[38] OpenZWave. `http://www.openzwave.com/`, March 2019, accessed: 2019-02-25.

[39] Z-Wave in Domoticz. `https://www.domoticz.com/wiki/Zwave`, 8 2018, accessed: 2019-02-25.

[40] Silicon Labs. *Z-Wave Application Security Layer (S0)*. March 2018.

[41] Abdullah, A. Advanced Encryption Standard (AES) Algorithm to Encrypt and Decrypt Data. June 2017.

[42] Fouladi, B.; Ghanoun, S. Security Evaluation of the Z-Wave Wireless Protocol. 2013.

[43] Silicon Labs. *Z-Wave Security Whitepaper*. March 2018.

[44] Diffie, W.; Hellman, M. New Directions in Cryptography. November 1976, pp. 644–654, doi:10.1109/WF-IoT.2018.8355224.

[45] Z-Shave. Exploiting Z-Wave downgrade attacks. `https://www.pentestpartners.com/security-blog/z-shave-exploiting-z-wave-downgrade-attacks/`, May 2018.

[46] ITU-T. Short range narrow-band digital radiocommunication transceivers – PHY and MAC layer specifications. Technical report, TELECOMMUNICATION STANDARDIZATION SECTOROF ITU, February 2012.

[47] Z-Wave Global Regions. `https://www.silabs.com/products/wireless/mesh-networking/z-wave/benefits/technology/global-regions/`, 2019.

[48] Silicon Labs. *Z-Wave Application Command Class Specification*. April 2019.

[49] Mitola, J. Software radios-survey, critical evaluation and future directions. In *[Proceedings] NTC-92: National Telesystems Conference*, May 1992, pp. 13/15–13/23, doi:10.1109/NTC.1992.267870.

[50] RTL-SDR Blog V3 Datasheet. Technical report, RTL-SDR, June 2017.

[51] Grove, B. Airspy SDR Receiver System. Technical report, Airspy, November 2015.

[52] HackRF One. `https://greatscottgadgets.com/hackrf/`, 2018, accessed: 2019-03-19.

[53] USRP B200/B210 Information. `https://files.ettus.com/b2x0_enclosure/`, 2018.

[54] SDR Showdown: HackRF vs. bladeRF vs. USRP. `http://www.taylorkillian.com/2013/08/sdr-showdown-hackrf-vs-bladerf-vs-usrp.html`, August 2013, accessed: 2019-03-19.

[55] Review: Airspy vs. SDRplay RSP vs. HackRF. February 2016: pp. 522–533, accessed: 2019-03-19.

[56] The BIG List of RTL-SDR Supported Software. https://www.rtl-sdr.com/big-list-rtl-sdr-supported-software/, February 2014, accessed: 2019-03-19.

[57] Csete, A. Gqrx SDR. http://gqrx.dk/, 2018, accessed: 2019-03-19.

[58] Lichtman, M. GNU Radio. https://wiki.gnuradio.org/index.php/Main_Page, March 2019, accessed: 2019-03-19.

[59] Markgraf, S.; Stolnikov, D.; et al. librtlsdr. https://github.com/librtlsdr/librtlsdr.

[60] Esbensen, A. rtl-zwave. https://github.com/andersesbensen/rtl-zwave, 2018.

[61] waving-z. https://github.com/baol/waving-z, 2015.

[62] Patel, M. *Implementation of FSK Modulation and Demodulationusing CD74HC4046A*. November 2013.

[63] Manchester Data Encoding for RadioCommunications. Technical report, Maxim Integrated Products, January 2005, accessed: 2019-03-20.

[64] Picod, J.-M.; Lebrun, A.; et al. waving-z. https://github.com/BastilleResearch/scapy-radio, 2016.

[65] Scapy. https://scapy.net/, 2019.

[66] EZ-Wave. https://github.com/cureHsu/EZ-Wave, accessed: 2019-04-27.

[67] Hall, J. L. A Practical Wireless Exploitation Framework for Z-Wave Networks. Technical report, Air Force Institute of Technology, March 2016.

[68] Seifert, C.; Welch, I.; et al. Taxonomy of Honeypots. 04 2006: pp. 3–8.

[69] The Google Hack Honeypot. http://ghh.sourceforge.net/, 2005, accessed: 2019-04-03.

[70] Developments of the Honeyd Virtual Honeypot. http://www.honeyd.org/, 2007, accessed: 2019-04-03.

[71] Project Honey Pot. https://www.projecthoneypot.org/about_us.php, 2019, accessed: 2019-04-03.

[72] Wang, M.; Santillan, J.; et al. ThingPot: an interactive Internet-of-Things honeypot. July 2018.

[73] Philips Hue. https://www2.meethue.com/en-us, 2019, accessed: 2019-04-04.

[74] HoneyThing. https://github.com/omererdem/honeything, 2016, accessed: 2019-04-03.

[75] Dowling, S.; Schukat, M.; et al. A ZigBee honeypot to assess IoT cyberattack behaviour. 06 2017, pp. 1–6, doi:10.1109/ISSC.2017.7983603.

[76] Pauna, A.; Bica, I.; et al. On the rewards of self-adaptive IoT honeypots. *Annals of Telecommunications*, 01 2019, doi:10.1007/s12243-018-0695-7.

[77] Semic, H.; Mrdovic, S. IoT honeypot: A multi-component solution for handling manual and Mirai-based attacks. 11 2017, pp. 1–4, doi:10.1109/TELFOR.2017.8249458.

[78] INSTEON. https://www.insteon.com/technology, note = Accessed: 2019-04-04.

[79] Wireshark. https://www.wireshark.org/, accessed: 2019-04-07.

[80] click. https://click.palletsprojects.com/en/7.x/, accessed: 2019-04-26.

[81] configparser. https://docs.python.org/3/library/configparser.html#module-configparser, accessed: 2019-04-26.

[82] multiprocessing. https://docs.python.org/2/library/multiprocessing.html, accessed: 2019-04-26.

[83] logging. https://docs.python.org/3/library/logging.html, accessed: 2019-04-27.

Appendix A

Acronyms

6loWPAN IPv6 over Low-Power Wireless Personal Area Networks

AMQP Advanced Message Queuing Protocol

BLE Bluetooth Low Energy

CRC Cyclic Redundancy Check

DAC Digital-to-analog

DSP Digital Signal Processing

GRC GNU Radio Companion

HTTP Hypertext Transfer Protocol

IoT Internet of Things

LAN Local Area Network

LPWAN Low-Power Wide-Area Network

MAC Medium Access Control

MQTT Message Queuing Telemetry Transport

PHY Physical

PPDU Physical protocol data units

PSDU Physical service data unit

RFID Radio Frequency Identification

SDR Software Defined Radio

Appendix B

User Manual

There are several tools, which needs to be prepared and installed before installation of the IoT honeypot.

B.1 Installation

1. RTL-SDR dongle

   RTL-SDR dongle is used for reception of Z-Wave frames. It is necessary to install drivers for this device. The installation guide which can be found in `https://www.rtl-sdr.com/rtl-sdr-quick-start-guide/`.

2. HackRF One

   HackRF One is used for transmission of Z-Wave frames. Installation guide for this device can be found in `https://github.com/mossmann/hackrf/wiki/Operating-System-Tips`

3. EZ-Wave

   Scapy-radio ensures digital signal processing and frame parsing. EZ-Wave offers examples of usage of Scapy-radio functions. Since EZ-Wave includes Scapy-radio too, Scapy-radio can be installed according to manual in `https://github.com/cureHsu/EZ-Wave`.

4. GNU Radio Companion

   GNU Radio Companion can be used for GRC block modification. Installation guide for GNU Radio Companion can be found in `https://wiki.gnuradio.org/index.php/InstallingGR`.

5. Block replacement

   It is highly recommended to replace files in

```
$HOME/.scapy/radio/Zwave/Zwave.grc
$HOME/.scapy/radio/Zwave/top_block.py
```

with files *Zwave.grc* and *top_block.py* that can be found on CD or Github repository of this thesis. These files are GRC blocks modified for usage with RTL-SDR dongle.

6. IoTpot

   This Python package is the final implementation and output of this thesis. Its implementation can be found on CD or Github repository of this thesis in `https://github.com/HalfDeadPie/IoTpot`. This Python package can be installed using the command:

```
python setup.py install
```

B.2 Usage

This section describes the usage of several subcommands of the IoT honeypot.

The option *–help* displays descriptions to all possible subcommands, options, and arguments that the package of the IoT honeypot offers:

```
iotpot --help
```

It is highly recommended to use a configuration file that can be accessed using option –config/-c. The example is:

```
iotpot --config config.cfg
```

All subcommands can be used within the main command. The example is:

```
iotpot --config config.cfg record
```

Appendix C

Contents of enclosed CD

- `readme.txt` ..... the file with CD contents description
- `src` ..... the directory of source codes
  - `grc-blocks` ..... Blocks for GNU Radio Companion
    - `Zwave.grc` .. Modified GNU Radio Companion blocks for Z-Wave
    - `top_block.py` ..... Generated block source code for Z-Wave
  - `IoT-Honeypot` ..... Python package
    - `iotpot` ..... implementation sources
    - `iotpot_data` ..... example of persistent data
    - `tests` ..... records used for tests
    - `LICENSE` ..... license of the work
    - `README.rst` ..... information about Python package
    - `setup.py` ..... the installation script of Python package
  - `thesis` ..... source code of the thesis
- `text` ..... the thesis text directory
  - `DP_Stefunko_Simon.pdf` ..... the thesis text in PDF format
  - `ZadanieDP_Stefunko_Simon.pdf` . the assignment text in PDF format