BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER SYSTEMS
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

SECURE GATEWAY FOR WIRELESS IOT PROTOCOLS

ZABEZPEČENÁ BRÁNA PRO BEZDRÁTOTOVÉ IOT PROTOKOLY

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **MARTIN HOŠALA**
AUTOR PRÁCE

SUPERVISOR **Doc. Ing. JAN KOŘENEK, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2019

Brno University of Technology
Faculty of Information Technology

Department of Computer Systems (DCSY) Academic year 2018/2019

Bachelor's Thesis Specification

22187

Student: **Hošala Martin**
Programme: Information Technology
Title: **Secure Gateway for Wireless IoT Protocols**
Category: Networking

Assignment:

1. Study widely used wireless IoT protocols and identify their security issues.
2. Get to know the BeeeOn IoT gateway and develop a security system to protect wireless IoT communication using existing detection systems.
3. Create a secure gateway that will run on Turris Omnia router and consist of BeeeOn Gateway and detection systems.
4. Design and implement a set of tests to check properties of the security system in the gateway.
5. Discuss created results and future improvements of the security system in the gateway.

Recommended literature:

- According to the supervisor's recommendation.

Requirements for the first semester:

- Items 1 to 3.

Detailed formal requirements can be found at http://www.fit.vutbr.cz/info/szz/

Supervisor: **Kořenek Jan, doc. Ing., Ph.D.**
Head of Department: Sekanina Lukáš, prof. Ing., Ph.D.
Beginning of work: November 1, 2018
Submission deadline: May 15, 2019
Approval date: May 9, 2019

Abstract

This work aimed to create a functional prototype of a secured gateway for wireless IoT protocols based on the BeeeOn IoT Gateway. To create the resulting solution, it was necessary to analyze existing IoT securing systems, propose their integration with the BeeeOn Gateway, and finally deploy the system. As the securing systems in this work, NEMEA modules developed within SIoT project were used. The resulting solution runs on Turris Omnia router and consists of the BeeeOn Gateway, five SIoT detection modules and other NEMEA modules necessary for detectors full functionality. Potential threats are being detected in the Z-Wave, BLE, and LoRaWAN networks. A user can interact with the system through a web interface of Coliot system, which is also a part of the resulting solution and serves to store and present detection results. The system functionality was verified experimentally and by a set of integration tests. The testing has revealed many deficiencies connected to used subsystems, and most of them were fixed. Resulting system is used within the SIoT project.

Abstrakt

Táto práca bola zameraná na vytvorenie funkčného prototypu zabezpečenej brány pre bezdrôtové IoT protokoly s využítím BeeeOn IoT Gateway. Na vytvorenie výsledného riešenia bolo potrebné analyzovať existujúce zabezpečovacie systémy IoT, navrhnúť ich integráciu s BeeeOn Gateway a systém nasadiť. Ako zabezpečovacie systémy v tejto práci boli použité moduly NEMEA vyvinuté v rámci projektu SIoT. Výsledné riešenie beží na routeri Turris Omnia a pozostáva z BeeeOn Gateway, piatich detekčných modulov SIoT a ďalších modulov NEMEA potrebných pre plnú funkčnosť detektorov. Potenciálne hrozby sa zisťujú v sieťach Z-Wave, BLE a LoRaWAN. Používateľ môže so systémom interagovať prostredníctvom webového rozhrania systému Coliot, ktorý je tiež súčasťou výsledného riešenia a slúži na ukladanie a prezentáciu výsledkov detekcie. Funkčnosť systému bola overená experimentálne a množinou integračných testov. Testovanie odhalilo mnoho nedostatkov spojených s použitými podsystémami a väčšina z nich bola opravená. Výsledný systém sa používa sa v rámci projektu SIoT.

Keywords

Internet of Things, Gateway, Smart Home, BeeeOn, Security, SIoT, NEMEA, Turris Omnia, LoRa, Z-Wave, BLE

Kľúčové slová

Internet vecí, Brána, Inteligentná domácnosť, BeeeOn, Bezpečnosť, SIoT, NEMEA, Turris Omnia, LoRa, Z-Wave, BLE

Reference

HOŠALA, Martin. *Secure Gateway for Wireless IoT Protocols.* Brno, 2019. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Doc. Ing. Jan Kořenek, Ph.D.

Rozšírený abstrakt

Trend Internetu vecí (skrátene IoT z angl. Internet of Things) sa rýchlo rozširuje a počet inteligentných krabičiek všade okolo nás každý deň rastie. Pre tento rast samozrejme existujú dobré dôvody. Správne používanie IoT zariadení môže napríklad zefektívniť výrobu a uľahčiť jej kontrolu, či zautomatizovať naše domácnosti. V neposlednom rade môže taktiež správna analýza dát z IoT zariadení viesť k užitočným predpovediam, ktoré môžu ďalej viesť k významným energetickým úsporám. Avšak výrobcovia IoT zariadení sú niekedy natoľko nadšený víziou zisku, že popri snahe čo najrýchlejšie priniesť inovatívne riešenia občas zabudnú dodržiavať základné bezpečnostné normy. To vedie k tomu, že IoT zariadenia sú často zraniteľné, podobne ako počítačové siete na začiatku 90. rokov.

Trend IoT nevynechal ani Fakultu informačných technológií Vysokého učení technického v Brne. Výskumná skupina akcelerovaných sieťových technológií[1] (skrátene ANT z angl. Accelerated Network Technologies) tu pracuje na IoT projekte s názvom BeeeOn[2]. Cieľom tohto projektu je vyvinúť open-source systém inteligentnej domácnosti, ktorý bude ľahko rozšíriteľný a bude podporovať široké spektrum koncových zariadení tretích strán. Výskumná skupina ANT tiež participuje na projekte s názvom SIoT[3], ktorého cieľom je zlepšiť bezpečnosť Internetu vecí tým, že preskúma jeho bezpečnostné slabiny a vynájde riešenia, ktoré zabránia ich využívaniu. Takýmito riešeniami sú typicky detekčné systémy, ktoré analyzujú IoT siete v reálnom čase, detegujú anomálie a tak identifikujú potenciálne bezpečnostné hrozby. V rámci projektu SIoT sa vyvíjajú detekčné systémy ako moduly pre systém NEMEA [4]. Správcom tohto systému je záujmové združenie CESNET[4].

Cieľom tejto práce bolo vytvoriť funkčný prototyp zabezpečenej brány pre bezdrôtové IoT protokoly s využítím BeeeOn IoT Gateway. V teoretickej časti práce sa nachádza popis fungovania IoT stietí ako takých a detailnejšia analýza niektorých často používaných bezdrôtových IoT protokolov s ich bezpečnostnými slabinami. Ďalej sa práca venuje popisu systémov BeeeOn Gateway, NEMEA a SIoT detekčných modulov.

Praktická časť práce sa venuje vytváraniu výsledného funkčného prototypu na základe poznatkov z teoretickej časti. Pre jeho vytvorenie bolo potrebné navrhnúť spôsob integrácie zabezpečovacích modulov s BeeeOn Gateway, a následne systém integrovať, nasadiť a otestovať. Ako zabezpečovacie systémy boli v tejto práci použité moduly systému NEMEA vyvinuté v rámci projektu SIoT. Výsledný systém je určený pre beh na routeri Turris Omnia a pozostáva z BeeeOn Gateway, piatich detekčných modulov SIoT a ďalších NEMEA modulov potrebných pre plnú funkčnosť detektorov. Potenciálne hrozby sa zisťujú v sieťach Z-Wave, BLE a LoRaWAN.

Používateľ môže s výsledným systémom interagovať prostredníctvom webového rozhrania systému Coliot, ktorý je tiež súčasťou výsledného riešenia a slúži na ukladanie a prezentáciu výsledkov detekcie. Coliot však nebeží na routeri, ale na virtuálnom počítači so systémom Ubuntu. Na tento virtuálny počítač by bolo takisto možné presunúť aj detektory. To by bolo výhodné v prípade, že by sa vyžadovalo nasadenie systému nejaký menej výkonný router. Funkčnosť systému bola overená experimentálne a množinou integračných testov. Testovanie odhalilo mnoho nedostatkov spojených s použitými podsystémami a väčšina z nich bola opravená. Systém bol takisto podrobený záťažovému testovaniu, ktoré ukázalo, že môže na routeri Turris Omnia bežať bez akýchkoľvek problémov. Výsledné riešenie sa využíva v rámci projektu SIoT.

[1] https://www.fit.vutbr.cz/research/groups/ant/index.php.cs
[2] https://www.beeeon.org/
[3] https://github.com/SecureGatewayIoT
[4] https://www.cesnet.cz /? lang = en

Secure Gateway for Wireless IoT Protocols

Declaration

Hereby I declare that this bachelor's thesis was prepared as an original author's work under the supervision of Doc. Ing. Jan Kořenek Ph.D. The supplementary information was provided by BeeeOn and SIoT project developers. All the relevant text information sources, which were used during preparation of this thesis, are properly cited and included in the list of references.

.....
Martin Hošala
May 22, 2019

Acknowledgements

First of all, I would like to thank God for making all this possible, encouraging me, and supporting me all the time. My thanks also belong to my supervisor Doc. Ing. Jan Kořenek Ph.D, who was always happy to help me and was patiently guiding me. Besides, I would like to thank the SIoT and BeeeOn projects developers. Namely Bc. David Bednařík, Ing. Ondřej Hujňák, RNDr. Radek Krejčí, Ing. Tomáš Čejka, Ing. Dominik Soukup, Ing. Erik Grešák, Ing. Jakub Jalowiczor, Bc. Jozef Halaj, Bc. Peter Tisovčík, and Bc. Klára Nečasová. Thank you all for your advice, valuable information, and your help in creating the resulting solution.

Contents

1 Introduction **2**

2 IoT networks and protocols **3**
2.1 Fog computing model 3
2.2 IoT gateway 6

3 Wireless IoT protocols **8**
3.1 Z-Wave 8
3.2 BLE 9
3.3 LoRaWAN 14

4 BeeeOn **18**
4.1 BeeeOn Gateway 19

5 System for Security Analysis **22**
5.1 Data Acquisition 24
5.2 Detectors 25

6 Gateway design **27**
6.1 System proposal 27

7 System integration **32**
7.1 Creation of tests 33
7.2 Nemea Collector incorporation 36
7.3 TurrisOS packages 36
7.4 Coliot 40
7.5 System deployment 40

8 Results **43**

9 Conclusion **46**

Bibliography **48**

Chapter 1

Introduction

The IoT trend is rapidly spreading, and the number of smart boxes everywhere around us is growing up every day. Of course, this growth has good reasons. The proper use of IoT devices can, for example, make it easier to control the factory and make production more efficient, make households fully automated, and last but not least, the proper analysis of IoT data can lead to useful predictions and therefore substantial energy saving. However, the vision of profit leads to the fact that the manufacturers are so excited about offering better IoT solutions, that they sometimes forget to adhere to the primary security standards. So, the IoT devices are often as vulnerable as computer networks of the early 1990s.

The IoT trend did not leave out the Faculty of Information Technology at the Brno University of Technology. Accelerated Network Technologies[1] (abbreviated ANT) research group works on a project named BeeeOn[2]. The goal of this project is to develop an open-source intelligent household system, which is easy-extendable and aims at broad coverage of the third-party end-devices support. The ANT research group also participates in a project called SIoT[3] that aims at improving the IoT security by examining its security vulnerabilities and inventing solutions to prevent their exploitation. These solutions are typically detection systems that serve for real-time IoT network analysis to detect traffic anomalies, and thus identify potential security threats. Within the SIoT project, detection systems are being developed as modules for the NEMEA system [4] that is maintained by the CESNET Association[4].

The goal of this work was to develop a secure gateway for wireless IoT protocols by integrating the BeeeOn's IoT gateway with the NEMEA system and its appropriate detection modules. The result of this integration is a functional sample of IoT gateway that is capable of detecting and reporting threats in the IoT network. In order to verify the achievement of the desired results, a test environment was created, whereby the functional sample of secure gateway was tested.

State of the art is described in four chapters. The first of them is dedicated to the general description of the IoT networks; the second one describes sensor layer IoT protocols appropriate for this work, and the other two chapters describe BeeeOn and NEMEA systems. The following chapter defines the solution requirements and design. The subsequent chapter describes the implementation of the solution and the last two chapters summarize and conclude the acquired results.

[1] https://www.fit.vutbr.cz/research/groups/ant/index.php.en
[2] https://www.beeeon.org/
[3] https://github.com/SecureGatewayIoT
[4] https://www.cesnet.cz/?lang=en

Chapter 2

IoT networks and protocols

Every day, more and more devices are connected to the Internet of Things (IoT). For 2025, the installed base of IoT devices is forecast to grow to more than 75 billion worldwide [17]. These billions of new devices generate an unprecedented volume of data and also represent countless new types of devices and data variations. Moving all these data to the cloud for analysis would require vast amounts of bandwidth, further by the time this movement and analysis would be completed, the opportunity to act might be gone [5]. Aforementioned shows that cloud models are not suitable for the volume, variety, and velocity of data that the IoT generates and a new model for analyzing and acting on these data is required.

2.1 Fog computing model

The model satisfying these requirements is called fog computing model [5]. The principle of this model stands in adding a few sub-layers to the communication, thus bringing the computing power closer to the end-devices.

The fog computing model extends the cloud model and brings the data processing physically closer to the devices that produce and act on IoT data. That is done using the devices with computing power, storage site, and Internet access, called **fog nodes**. Any device that fulfills given requirements can become a fog node. Examples of such devices include routers, switches, industrial controllers, or embedded servers. The fog computing model consists of three layers: the end-devices layer, the fog layer, and the cloud layer. The basic idea of the fog computing model is displayed in figure 2.1.

The fog nodes, in addition to their standard functionality, provide a part of their resources for running external fog applications. The general purpose of the fog applications is monitoring and analyzing real-time data from network-connected devices and initiating actions, such as locking the door or sending an alert to a user.

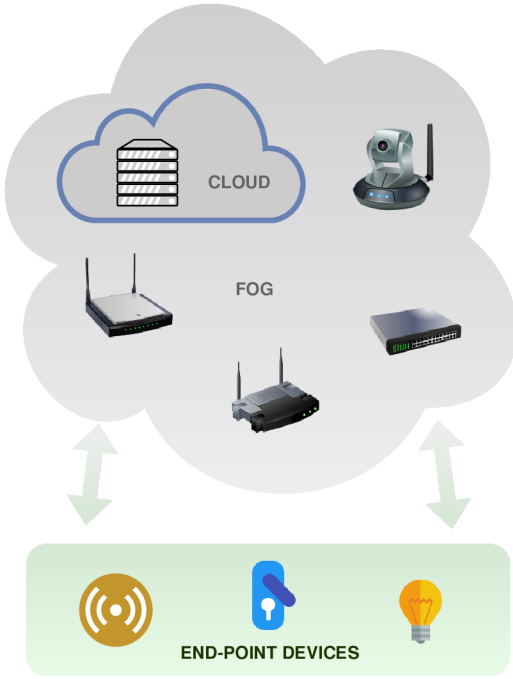

Figure 2.1: Fog computing model

The fog nodes closest to the network edge ingest the data from IoT gateways (fog node can also be the IoT gateway itself), which are directly connected to the IoT devices. Then the fog application directs the different type of data to an optimal place for their analysis. The most time-sensitive data is analyzed on the fog node closest to the devices generating the data. Data that processing can wait a few seconds or minutes are passed along to an aggregation node which analyses them and initiates appropriate action. Data that are less time sensitive are sent to the cloud for historical analysis, big data analytics, and long-term storage. The usage of the fog computing model comes along with the following benefits:

- **Lower time and bandwidth expenses:**
  Selected data are processed locally instead of being sent to the cloud. That conserves the network bandwidth and also lowers the latency time, which can be crucial when a critical action, such as opening a valve in response to a pressure reading, should be invoked.

- **Security and privacy enhancement:**
  Sensitive data do not need to be sent across the internet to the cloud for processing, so the chances of their leakage are lowered. Further, the fog nodes are typically non-stop connected to the internet and the power supply, so chances of the non-processing the time-sensitive data due to a connection failure are also minimized.

- **Shading of the end-devices diversity:**
  Thanks to the fog nodes, the end-devices can communicate almost directly and while still using different communication protocols. Also, the network administration is simplified, as long as it is not needed to access the end-devices every time, but most of the time, managing the appropriate fog layers is enough.

Above mentioned shows that the Fog Computing Model came along with many benefits and was invented directly for the IoT networks and also, as long as it only describes the IoT network structure and the data processing distribution, it does not contain weak security spots itself. However, using the Fog Computing Model principles, the IoT communication model can be divided into three layers [16], where each contains specific vulnerabilities:

1. **Sensor Layer**
   The sensor layer includes all end-devices that obtain information from their surroundings or perform the desired actions. Many end-devices can be connected to one IoT gateway, while also using various communication protocols. The principle of communication and the capabilities of the topology differ according to the technology used.

   The significant vulnerabilities of this layer are, in particular, wireless protocols. If no security features are used, communication via wireless protocols can easily be intercepted or modified. Also, there may be devices that are marked as secure, but still uses an older version of the communications protocols that come along with out-of-date security features or contain implementation errors. This case is dangerous because it raises a false sense of security. Attacks can also target battery-powered items that can be discharged by excessive traffic.

2. **Network Layer**
   To the network layer, the sensor layer is connected via an IoT gateway. The communication on the network layer typically uses the TCP/IP protocol suite and provides the transmission of obtained sensor data from the IoT gateway to other services. Also, the remote management of the gateway, and thus also the remote management of the end-devices takes place on this layer. In most cases, the connection is created using HTTPS (Hypertext Transfer Protocol Secure) or VPN (Virtual Private Network). However, the often used protocols also include MQTT (Message Queuing Telemetry Transport), COAP (Constrained Application Protocol) or AMQP (Advanced Message Queuing Protocol).

   It follows from the above mentioned that as long as the network layer uses traditional TCP/IP communication, its security threats are identical to the traditional network threats. Thereby, the standard principles of trust, integrity, and accessibility must be respected to prevent traditional attacks such as DDoS (Distributed Denial Of Service), MITM (Man In The Middle) or information falsification. The main difference of this layer and the traditional network is that the most common communication on this layer is M2M (Machine To Machine) communication, so it is necessary to use appropriate communication interfaces and to update the devices whenever possible. An outdated version of the firmware can leave the door open to the attack which can cause unusual behavior or even complete IoT network control takeover.

3. **Application Layer**
   Finally, the application layer takes care of data processing and their long-term storage. It also communicates with the users and allows them to configure the entire network. However, as the IoT network can consist of thousands of devices, it is critical for its topology management to support automation. This layer is typically located in the data center, provides remote access, and its security threats can be likened to cloud computing security threats. The complexity of the application layer itself, and

therefore also the complexity of its security, depends on the required functionality and can vary from application to application in significant ways. Common attacks threatening this layer include Buffer Overflow, SQL Injection, or DDoS.

In general, the problem is also the way of an IoT network deployment. IoT end-devices have different parameters than conventional user devices, and their communication also differs as long as it is less heterogeneous and often based on M2M. The deployment of many end-devices into one segment complicates security rules and increases the impact of a potential attack, as it is often possible to extend the control takeover or the data falsification from a single successfully attacked device to other nodes in the same segment.

2.2 IoT gateway

In the fog computing model, the IoT gateway [16, 15] is a physical device or software program that serves as the connection point between the end-devices layer and the fog layer. This is shown in figure 2.2. IoT gateway nodes can also be referred to as Edge nodes, and they are typically computationally more powerful than the end-devices. For the IoT networks, it is typical to consist of many different types of end-devices which use different communication protocols. The main purpose of the IoT gateway is to collect data from these devices. Because of their diversity, IoT gateway needs to include additional communication interfaces that allow it to communicate via all sorts of wireless and also wired communication protocols. Examples include Z-Wave, BLE, or Zigbee. More wireless IoT protocols and their detailed description can be found in the below sections. Collected data is then provided to the higher layers for the processing. If the IoT gateway is powerful enough, it can also be a fog node itself. In that case, the IoT gateway can directly process some of the collected data.

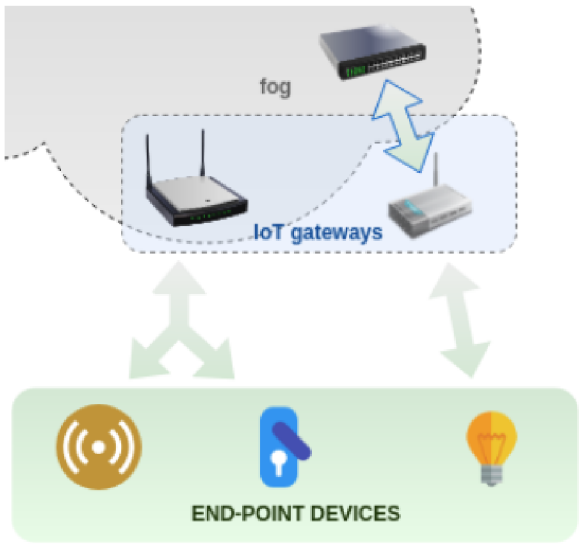

Figure 2.2: IoT gateways

Currently, there are many different IoT gateways available. Their parameters vary according to deployment and operational requirements. A significant issue in this area is the low emphasis on security features that enable remote gateway control, traffic control,

and software upgrades. An IoT gateway that is mainly focused on the security and broad IoT protocols coverage is being developed at BUT[1]'s Faculty of Information Technology within the project called BeeeOn. The mentioned IoT gateway is described in section 4.1.

[1] https://www.vutbr.cz/en/

Chapter 3

Wireless IoT protocols

3.1 Z-Wave

Z-Wave [21, 8] is a proprietary wireless communication protocol primarily designed for home automation. It is owned and maintained by the Z-Wave alliance[1], which provides development licenses, protocol documentation, and also certifies all the Z-Wave components. Z-Wave was introduced in 2001 and later in 2013 Z-Wave alliance started a new certification program called `Z-Wave Plus`. Z-Wave Plus certified devices are backward compatible with devices with previous Z-Wave certificates but must follow conditions that bring many user-friendly orientated benefits. The most significant advantages of the Z-Wave are that it is widely spread protocol in the field of home automation, Z-Wave devices are all fully inter-operable [22] and affordable.

Z-Wave transmits messages using low-frequency (`800-900MHz`) radio waves. The Z-Wave communication speed ranges from 9.6kb/s to 40kb/s, and its range can reach up to 100m outdoors. Devices which participate in a Z-Wave network are called Z-Wave nodes, and each of them serves as a controller or as a slave. Multiple controllers can participate in one Z-Wave network, and each of them is allowed to control all the slave nodes within the network. However, one of them must be primary. At a given time, one primary controller can manage one Z-Wave network. Such network is then identified by a unique 32-bit `Home ID` which value is written to the controller's Z-Wave chip by the manufacturer and cannot be changed by the controller software [6]. The significant advantage of the Z-Wave networks is that it can span much farther than the radio range of a single device. It is possible as the Z-Wave devices serve also like the routers and the communication between the devices within one network does not need to be direct. So, theoretically, to be the part of a Z-Wave network, it is enough for the device to be placed in the radio range of one another device if it is possible to reach a controller device via it. That implies a source-routed mesh topology of the Z-Wave networks. The message can hop via up to four nodes, while the real distance between two communicating nodes can typically be up to 30–40 meters. However, to become a part of the Z-Wave network, the new node needs to be paired with the primary controller, and this procedure demands a direct connection. Furthermore, the same applies to the unpairing process.

Within the Z-Wave network, each node is identified by an 8-bit identifier called `Node ID`. Thus, each Z-Wave node that is in use can be uniquely identified by the pair `Home ID` and `Node ID`. A new network participator obtains its Node ID during the pairing procedure.

[1] https://z-wavealliance.org/

The `Node ID` of a primary controller always 1. Nevertheless, the short length of the Node ID limitate the network size – a Z-wave network can consist of only up to 232 devices. However, it is possible to bridge multiple networks if further devices are needed.

Each Z-Wave node includes the definition of the supported Command Classes. These define the commands that the appropriate node can understand and the form of the node's response to them. During the pairing process, the new node sends its Command Classes to the controller. Every Z-Wave message needs to be acknowledged. If the acknowledgment message does not arrive, the original message is resent up to 3 times, then canceled.

3.1.1 Security and vulnerabilities

Since 2009 Z-Wave used a security standard called S0. This standard uses 128-bit AES encryption[2] to secure the communication. Despite, in the S0 standard, the encryption is not required but optional. If it is used, the S0 network key exchange takes place during a pairing procedure. However, this exchange is encrypted with a fixed key of all zeros. Consequently, the attacker can obtain the network key while a new device is being paired and then attack any device within the network. Therefore it is recommended to pair devices in a secure environment [8].

Significant advances in Z-Wave security occurred in 2016 with the arrival of the new Z-Wave security standard named `Security 2` (abbreviated `S2`). This standard came along with mandatory message encryption and the ECDH algorithm usage for the keys exchange. Since April 2017, devices need to meet the `S2` standard to be Z-Wave certified. Although the S2 security enhancement solved all known Z-Wave vulnerabilities, there is still a majority of devices without it [22]. The encryption of communication is entirely optional for these devices, and it is not commonly used. Thus their communication can be easily intercepted and modified.

Furthermore, the Z-Wave devices are backward compatible, and an S2 device pairs as the S0, when pairing with an S0 device. The information about the device security level is exchanged unencrypted at the beginning of the pairing process. Therefore, an active attacker, present at the time of pairing, can downgrade an S2 pairing to S0, and thereby make it possible to intercept and inject S0 traffic on the Z-Wave network [18].

3.2 BLE

Bluetooth Low Energy (colloquially BLE, formerly Bluetooth Smart) is a wireless network technology designed and maintained by the Bluetooth SIG[3] intended for IoT networks such as intelligent houses. Its specification, accompanying the Classic Bluetooth and Bluetooth high speed, is a part of the Bluetooth 4.0 specification, also known as Bluetooth Smart, introduced in 2010. However, significant BLE improvement came along with the specification of the Bluetooth 5.0. Transmission range of BLE is similar to the range of Classic Bluetooth, and these two protocols also operate in the same spectrum range. Nevertheless, BLE aims at the reduction of power consumption, and it is not compatible with the Classic Bluetooth. Information in this chapter are based on [8, 7, 11, 19, 9].

BLE operates in the spectrum range 2.400-2.4835 GHz and uses 40 2-MHz channels. Although the Classic Bluetooth operates in the same spectrum range, it uses 79 1-MHz channels, and that is the main reason for the BLE and Classic Bluetooth incompatibility.

[2] https://en.wikipedia.org/wiki/Advanced_Encryption_Standard
[3] https://www.bluetooth.com/

The maximum data transmission speed of BLE 4.0 is 1 Mbit/s, and its range can reach up to 100m in a suitable environment. BLE 5.0 can operate in two transfer modes. The first mode aims at the communication speed and allows data transmission speed to reach 2 Mbit/s. On the other hand, the second mode aims at the transmission range, which can span up to 1000m in ideal condition. However, the transmission speed of the second mode is the same if not even slower compared to BLE 4.0.

The BLE networks consist of two device types – Masters and Slaves. The term Slave is used for end-devices and the term Master for their controllers. In the BLE version 4.0, an end-device could only be connected to one controller, and that led to the star topology of the BLE networks, as shown in figure 3.1. It changed in 2017 with the arrival of the Bluetooth 5.0 Specification that came along with Bluetooth Mesh Networking Specifications which allowed building BLE networks with the mesh topology, which is also displayed in the figure 3.2. Each Bluetooth and also BLE device is identified by a globally unique 48-bit Bluetooth device address (BD_ADDR).

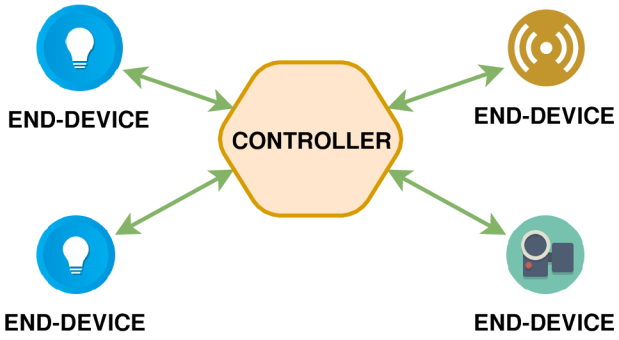

Figure 3.1: BLE4.0 star topology

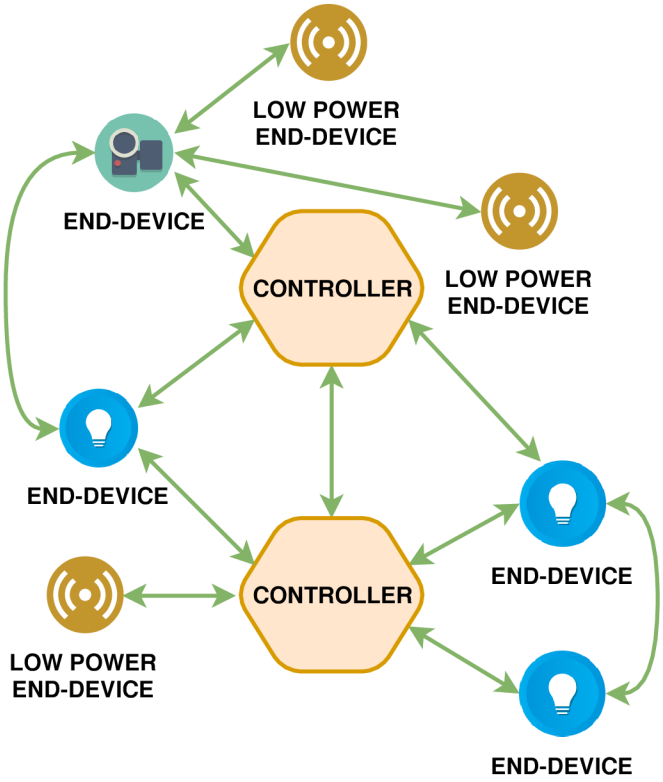

Figure 3.2: BLE5.0 mesh topology

The communication between the end-device and the controller device typically consists of five steps [7]:

1. **Transmission of a connection request**
   For device discovery purposes, there are three BLE channels reserved. When a pairing mode is initiated at an end-device, it starts to broadcast so-called advertisement packets on at least one of these channels. These advertisements include basic device information such as supported services, device name, vendor name, and they are repetitively sent until the connection is initiated or the timeout is passed.

2. **Receivement of a connection request**
   When a controller is in scan mode, it periodically listens on advertisement channels for a duration called scan window. Then the controller offers a user a list of all end-devices, from which it received a message.

3. **Initation of a connection**
   Once the user chooses an end-device to connect with, the controller stops the advertisement scanning and initiates a connection.

4. **Connection set up**
   During this phase, the controller scans end-device for the available services, and both parties agree on communication parameters.

5. **Main communication**
   The controller exchanges information with the end-device using appropriate (read, write, or notify) requests and responses.

This communication model is captured in figure 3.3, which displays the time course and communication direction of BLE connection steps as mentioned above. However, depending on the usage scenario, the direct connection set up is not always necessary, and in some cases, the controller can handle advertisements without it.

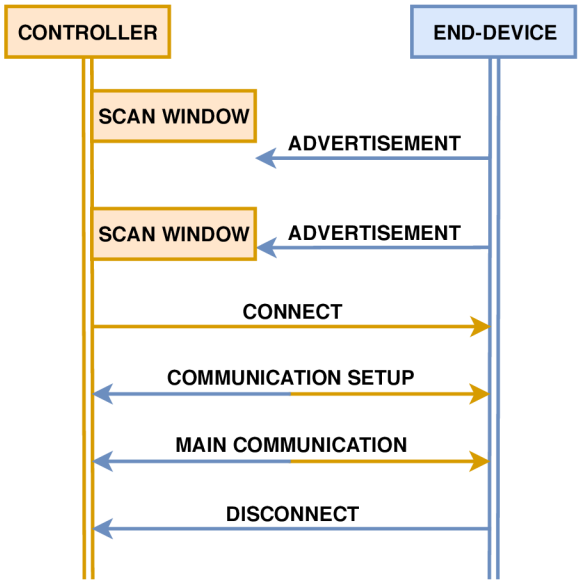

Figure 3.3: BLE communication model

3.2.1 Security and vulnerabilities

A potential attack on the BLE communication is complicated by the fact, that the transmission does not take place on constant frequency, but hops from channel to channel. However, there are several nonexpensive options to accomplish communication interception, including open-source platform Ubertooth[4]. Information about BLE Security and vulnerabilities are mainly based on [7, 3, 8, 14].

To avoid interception of BLE traffic, BLE uses highly secure 128-bit AES encryption operating in CCM mode. However, to establish a secured connection, the devices must first go through the pairing procedure. The process, and thus also the security of the pairing procedure differs depending on the BLE version. The devices with a BLE version older than 4.2 use a custom key exchange protocol unique to BLE and their pairing procedure is known as Legacy Pairing. During this procedure, the devices exchange via one of the pairing methods so-called Temporary Key (TK) which is then used to derive a Short Term Key (STK) that serves to encrypt the connection. If it is required to create a bond, a master device sets up a Long Term Key (LTK) that would be used to secure future sessions, and transmit it to the slave. The pairing procedure of the BLE 4.2, and also the later versions, is known as Secure Pairing and it uses, instead of the formerly used custom one, the Elliptic-curve Diffie-Hellman (ECDH) key exchange protocol. Both devices which are going through Secure Pairing procedure generate an ECDH public-private key pair. The public keys are then exchanged and used to compute a Diffie-Hellman key. The connection is then authenticated using one of the pairing methods, and when everything is correct, a

[4]https://github.com/greatscottgadgets/ubertooth

Long Term Key is derived from the Diffie-Hellman key, and then it is used to encrypt the communication.

The key exchange in the Legacy Pairing is possible by three methods. The Secure Pairing improved all of them and introduced an additional one. These methods are:

Just Works The main advantage of this method is that it proceeds automatically; the user does not need to confirm anything, and thus, the device does not need to have a display. Therefore, as long as many BLE devices do not have a display, this method is the most commonly used one.

In the Legacy Pairing, the Just Works means that the TK is 0 and thus, with the usage of a brute force, it is easy for a potential attacker to find out the STK and decrypt the communication. Moreover, this method offers no way of verifying the devices that participate in the connection, and thus, there is no MITM attack protection.

When Just Works method is used in the Secure Pairing, devices first exchange the public keys and compute Diffie-Hellman keys. After that, the initiating device generates a random nonce and uses it with Diffie-Hellman key to generate a confirmation value (Cb). Then both the nonce and the Cb are sent to the non-initiating device, which generates confirmation value (Ca) from its Diffie-Hellman key and the received nonce. If Ca matches with Cb, both devices computed the same Diffie-Hellman keys and the connection can proceed.

Thanks to the ECDH, the Just Works in Secure Pairing is markedly more resistant to passive eavesdropping than in Legacy pairing. Nevertheless, the communication participators still cannot be verified, and thus, the MITM vulnerability stays the same.

Passkey If this method is used in the Legacy Pairing, the TK is a 6-digit number which needs to be passed between the devices by a user. Devices can use different mechanisms to exchange TK. For example, one device can generate a number and show it to the user, who has to enter the number to the other device. As long as a potential attacker does not intercept the pairing procedure, the communication will be protected against passive eavesdropping. However, with the sniffed passkey value, the attacker can use brute force to derive STK just as when the Just Works method is used. This method is also generally considered to be MITM resistant, yet Tomas Rosa presented a way of bypassing passkey authentification in his whitepaper [13].

In the Secure Pairing, the user needs to input an equal number to each device. The authentication then proceeds like in the Just Works method, but the confirmation values are calculated also using a bit from the input passkey, and the checking process repeats until each bit is checked.

Out of Band (OOB) In the OOB pairing, the appropriate information is exchanged using a different communication channel (wireless technology, typically NFC). This information includes TK in the Legacy pairing and the public keys, nonces, and confirmation values in the Secure Pairing. If the used channel is immune against MITM and eavesdropping, the BLE connection will be secured. In the Legacy Pairing, this is the most secure method, yet it is not widely adopted.

Numeric Comparison Numeric Comparison came along with the BLE 4.2; thus, it is only available in the Secure Pairing. In this method, the pairing proceeds the same

as in the Just Works method. The thing is, that after the connection is set up, both devices generate 6-digit code and let a user check their match. Usage of this method secures the connection against MITM attacks [16].

As shown above, the BLE pairing procedure cannot be considered as secure until version 4.2. With advanced knowledge of BLE protocol, a potential attacker can decrypt the communication and use man in the middle attack. Even though mostly used pairing methods are susceptible to passive interception, the idea is that it is supposed to be performed only once and in a secure environment. Thus it is meaningful to pair devices that use elder protocol versions in a secure area. Nevertheless, there is also a way to force already paired devices to renew their Long Term Key and thus re-launch the pairing process. That can be accomplished by injecting the appropriate link layer message (LL_REJECT_IND) at the proper moment during the session initialization.

Even though later versions improve the pairing process security, the encryption remains optional, and many vendors do not implement it. Some vendors implement encryption and other security features on the application layer, but their implementations commonly contain flaws or weak protection against specific threats (MITM or replay attacks). Another problem related to BLE security is the complexity of the BLE specification itself. Its vastness leads to BLE implementation inaccuracies, and despite the usage of all the available security features, there can still be vulnerabilities caused by the implementation.

3.3 LoRaWAN

LoRaWAN is an IoT protocol maintained by the LoRa Alliance[5], based on their proprietary LoRa (Long Range) radio technology. It is the next example of low-power wireless protocols designed for cheap and secure communication in the IoT networks. Its most significant advantage is its range as in a suitable environment it can reach up to 20km. LoRaWAN communication takes place on the sub-gigahertz license-free bands, and its data rate ranges from 0.3 kbit/s to 50 kbit/s. Information about LoRaWAN are based on [10, 8].

The sensor layer of LoRaWAN networks consists of two device types – gateways and end-devices. The gateways communicate with the end-devices using LoRa radio technology and with the central Network Server using standard TCP/IP communication. Consequently, they serve as bridges connecting the end-devices with the central Network Server. On the other hand, end-devices do not route messages and cannot communicate with each other; they serve only for their particular purposes. One end-device can be connected to multiple gateways, but all these gateways must be connected with the same Network Server. That leads to a star-of-stars topology of the LoRaWAN network, which is also displayed in figure 3.4. The Network Server routes all messages received from the gateway to an appropriate Application Server that proceeds them, or to the Join Server that handles the Join procedures and the session keys derivation. Every end-device has its Device Identifier (DevEUI) that is a 64-bit unique identifier set by vendors or developers. Additionally, all end-devices within the network have a 32-bit address allocated by the Network Server, called End-device Address (DevAddr), that serves for the communication addressing.

[5] https://lora-alliance.org/

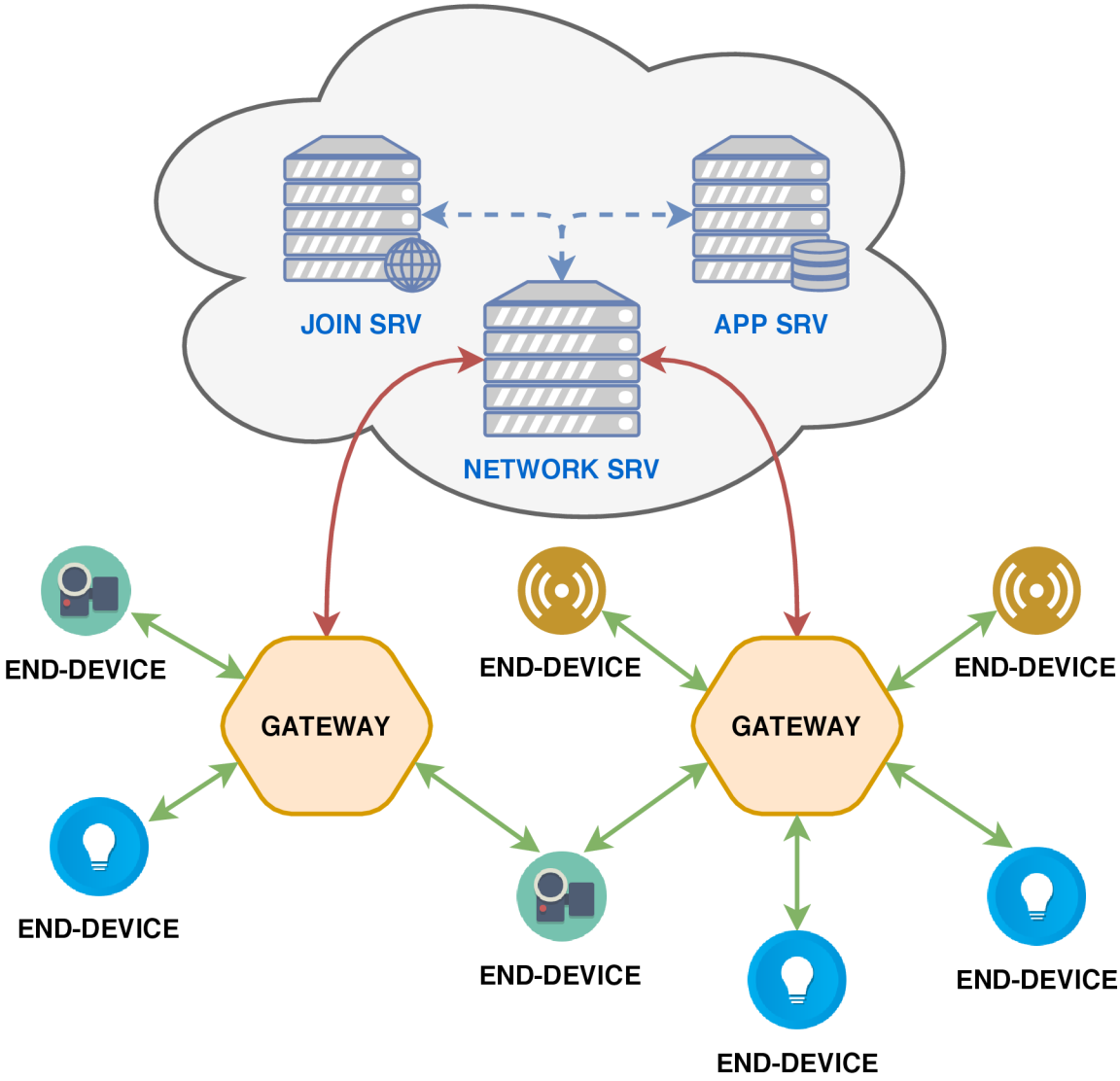

Figure 3.4: LoRaWAN network topology

The end-device communication model is divided into three classes:

Class A - All end-devices: End-devices support basic bi-directional communication with a gateway. The messages from the end-devices can be transmitted at any time, but they listen for incoming messages only in the two reception windows at a specified time after the transmission. Then they switch to the power-saving mode and do not listen to any messages until their next transmission. All end-devices must implement the Class A features.

Class B - Beacon: Class B option serves to have end-devices that are available for reception at a predictable time and thus can also be reached at another time, not only after their transmission. The end-devices listen to the beacon messages, that are regularly sent by the gateway and serve to synchronize all end-devices in the network to open

a short additional reception window (called “ping slot”) periodically at an expected time.

Class C - Continuously listening: Finally, the duration of the reception windows of the end-devices implanting the Class C option is maximized, so they listen almost continuously for all messages. The reception windows are closed only at the time when the end-devices transmit messages. This approach is used for devices which have a sufficient energy source and do not need to minimize the reception time to save power.

An end-device can implement multiple classes mentioned above and switch from one class to another. Figure 3.5 shows the reception windows opening principle for each class. Also, in the LoRaWAN network, end-devices using different classes can coexist, but there is no particular message type specified by the LoRaWAN protocol to inform the gateway about the device class. If this information is needed, its obtainment must be handled on the application layer.

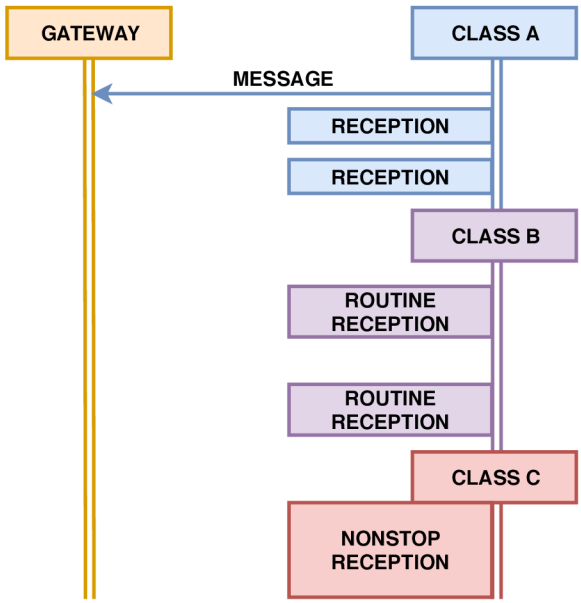

Figure 3.5: LoRaWAN end-devices reception windows

3.3.1 Security and vulnerabilities

LoRaWAN uses 128-bit AES encryption operating in CTR mode to secure the communication on the network layer and also on the application layer. Two 128-bit keys called NwkKey and AppKey are stored in the end-device and are used during the joining procedure to derive end-device specific session keys. The joining procedure can be realized in two ways. First of them is called ABP (Activation by Personalization) and the keys exchange does not take place in it, because the session keys are generated at production time, stored in the end-device and locked to a specific deployment. Security analysis of the LoRaWAN is based on [10, 1, 8].

The second one is called OTAA (Over the Air Activation), and this procedure is initiated by a Join-request message sent by an end-device. This message contains DevEUI, JoinEUI

which is a unique 64-bit identifier of a Join Server and the DevNonce which is a counter incremented with every sent Join-request message. The Join Server keeps track of all received DevEUI values and appropriates DevNonce values to prevent a replay attack. However, the Join-request message is not encrypted, and if eavesdropped, it can still be later used for a replay attack, as shown in. If the end-device is permitted to join a network, the Join Server response with a Join-accept message that consists of JoinNonce, DevAddr, and other network and configuration information. Sending the Join-accept message also causes incrementation of the JoinNonce, which is a counter specific for a Join Server. Both the DevNonce and JoinNocne participate in the session keys derivation, although the exact way and other participators depend on the LoRaWAN version.

To prevent a replay attack of messages from regular communication, LoRaWAN uses message counters to generate different keystream for each message. This keystream is then used to encrypt the message payload, which is done using XOR operation with an appropriate key from keystream. However, it is also typical for the end-devices to reboot or reset after some. In that case, message counters are zeroed, as the whole pairing procedure must be done again. As devices always use the same session keys in the ABP mode, it is possible to use a replay attack. A potential attacker can capture messages with a specific sequence number and inject them later, after the end-device restarts. The Network Server will then ignore the message containing information that counters had been zeroed. Therefore, following messages sent from the end-device would be considered to have a wrong sequence number and thus also ignored. That would lead to DoS of the end-device until the message counter reaches the value stored on the server [1].

The further vulnerability of the LoRaWAN protocol is beacon messages that serve to synchronize reception windows on end-devices implementing Class B. Beacon messages are not encrypted, nor signed, and contain data that can be misused. These data include, for example, GPS coordinates of the gateway, which makes it possible to localize and physically attack the gateway. Moreover, it is possible for an attacker to inject malicious beacon messages that contain the wrong synchronization time and thus cause the DoS of the end-devices as they will open reception widows at the incorrect time. Besides, by injecting numerous malicious beacon messages, an attacker can drain the battery of battery powered end-devices. When an end-device receives multiple beacon messages containing always different synchronization time, it discharges its battery faster, as it has to process these messages and always re-synchronize its reception windows.

Another security flaw is based on the fact that in the ABP mode, end-device uses the same encryption keys for each session. This fact makes it possible to retrieve the content of the eavesdropped messages, although they are encrypted. To do so, an attacker needs to eavesdrop the communication of an end-device, then wait for its restart and eavesdrop again. Captured messages from two sessions can be then paired based on their sequence numbers to get message pairs containing messages that were encrypted using the same keystream. Since the encryption of the messages in the CTR mode is done using the XOR operation with the appropriate keystream, it is possible to do the XOR operation of two eavesdropped messages encrypted using the same keystream and get the result that is equal to the result of the XOR operation of their non-encrypted forms. With a knowledge of the LoRaWAN communication details, an attacker can then reconstruct these messages and thus get their plaintext. This attack is described in more detail in [20].

Chapter 4

BeeeOn

The goal of this work is to develop a functional sample of the secure IoT gateway for wireless IoT protocols based on BeeeOn Gateway (BGW). The BGW is being developed within an open-source and open-hardware project called BeeeOn[1]. This project aims to develop a modular, easily expandable, and secure system for the intelligent household. It is maintained by Accelerated Network Technologies Research Group operating[2] at the Faculty of Information Technology in Brno.

The BeeeOn system consists of several layers, as shown in 4.1, and endeavors to cover as much third-party end-devices available on the market as possible. These are sensors and actuators of various kinds, communicating via different protocols.

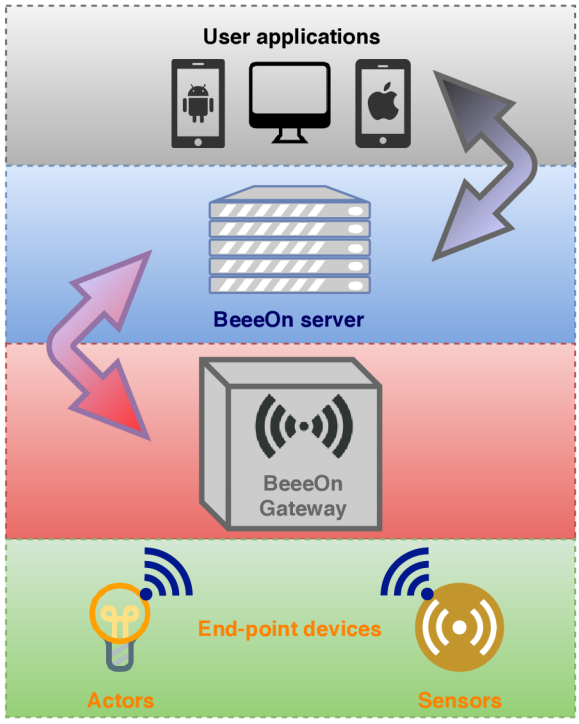

Figure 4.1: Layers of the BeeeOn system

[1]https://www.beeeon.org/
[2]https://www.fit.vutbr.cz/research/groups/ant/index.php.en

Sensors are used as a source of information. They can measure physical quantities or detect door openness. Actuators are devices that are manageable and able to change state. An example of an actuator device is a smart socket that can be switched on or off at a distance or a smart light bulb that can be remotely adjusted for the intensity or color of the light. Both the actuators and the sensors are wirelessly connected to the central system unit – Gateway.

Gateway [2] is the center of the BeeeOn Intelligent Household. It is a platform-independent system. However, the development is affected by the fact that the standard platform for Gateway is a specific microcomputer with ARM architecture and GNU/Linux operating system[3]. This microcomputer can be expanded by hardware modules (USB dongles, SPI peripherals, and so on) to include a variety of wireless IoT interfaces. Besides its main functionality, the gateway provides an interface for real-time operational details monitoring.

The BeeeOn system also consists of BeeeOn Server and user applications. The BeeeOn Server is mainly responsible for user accounts and data management, while user applications are used to control and monitor the household. However, these parts of the BeeeOn system are not relevant to this work.

4.1 BeeeOn Gateway

BeeeOn Gateway (BGW) [2] is a modular system whose main tasks are connecting and management of end-devices and communication with the server. Communication with the end-devices takes place via wireless protocols, and at the time of writing, BGW supports WiFi, Z-Wave, BLE, and IQRF protocol. BGW source codes are written in the C++ language C++11, with significant usage of the POCO[4] libraries. The basic structure of the BGW is shown in Figure 4.2 and consists of five components.

[3] http://nanopi.io/nanopi-neo.html
[4] https://pocoproject.org/

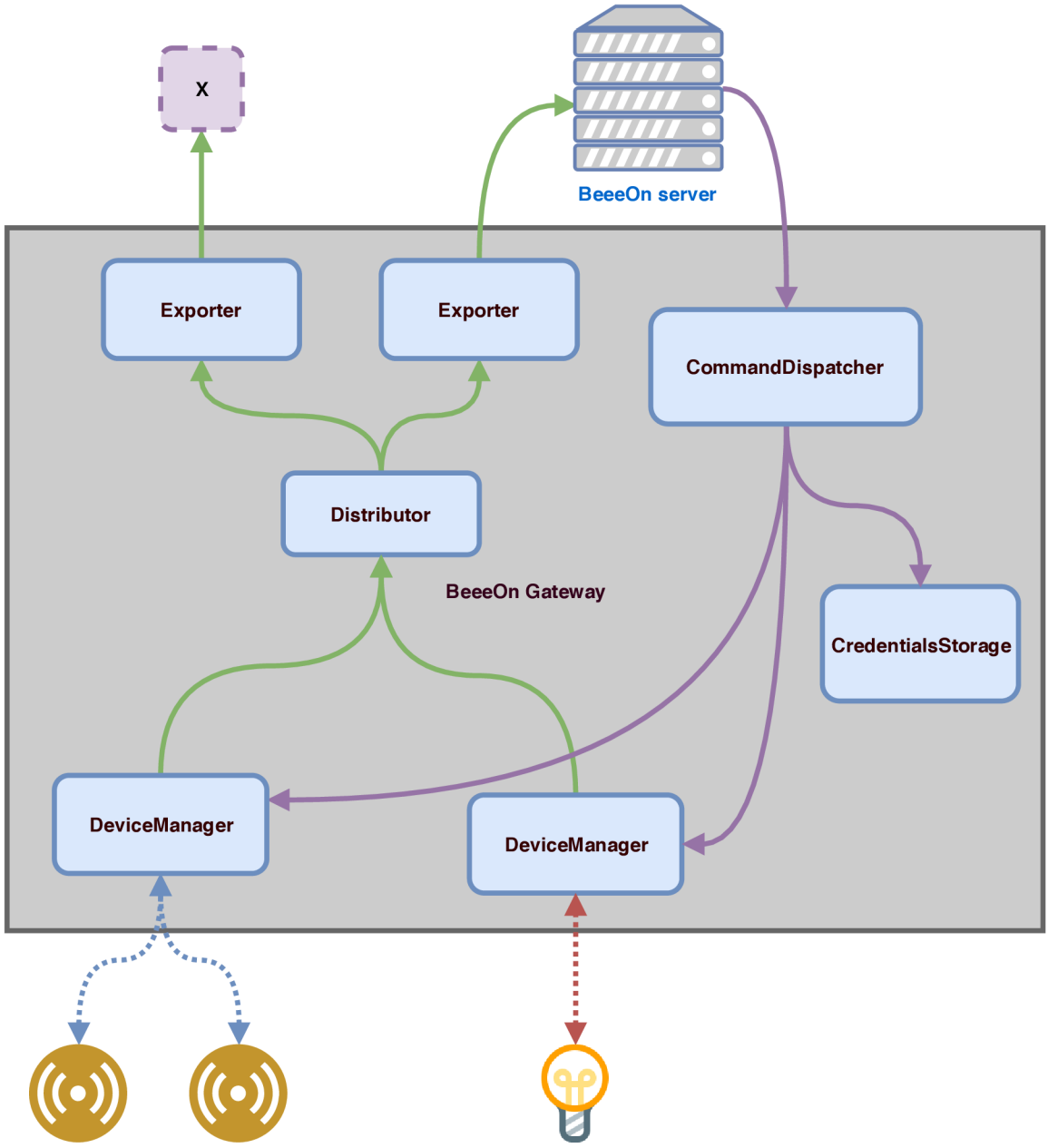

Figure 4.2: BeeeOn Gateway structure

DeviceManager serves to communicate, pair, and control a group of end-devices that communicate similarly. The data from the end-devices received by appropriate DeviceManager are converted to the uniform format and sent to the Distributor for further processing.

CommandDispatcher is responsible for sending commands to end-devices and also for the Gateway itself. Command examples include a command to turn on the Gateway pairing mode, an end-device shutdown command, or an order to change the status of an actuator.

CredentialsStorage is used to store credentials needed to authenticate devices connected to Gateway.

Distributor cooperates with Exporters and forwards the received data to them. However, the Distributor can also implement protection from the potential data loss due to adverse events such as the internet inaccessibility.

Exporter is responsible for delivering the data in an implementation-specific way to the final destination. For example, it can deliver the data using the MQTT protocol or export them to the BeeeOn server.

Also, each of these components uses the Observer Design Pattern and defines appropriate observable events to provide information about its current state. Therefore it is possible to implement a specific component, a so-called collector, as shown in Figure 4.3. The collector implements the observer role and observes demanded events. Therefore it gets a notification with related data when an event occurs. The exporter collects and exports this data for further processing. The data can be then analyzed to create statistics or to detect network anomalies and help identify potential threats.

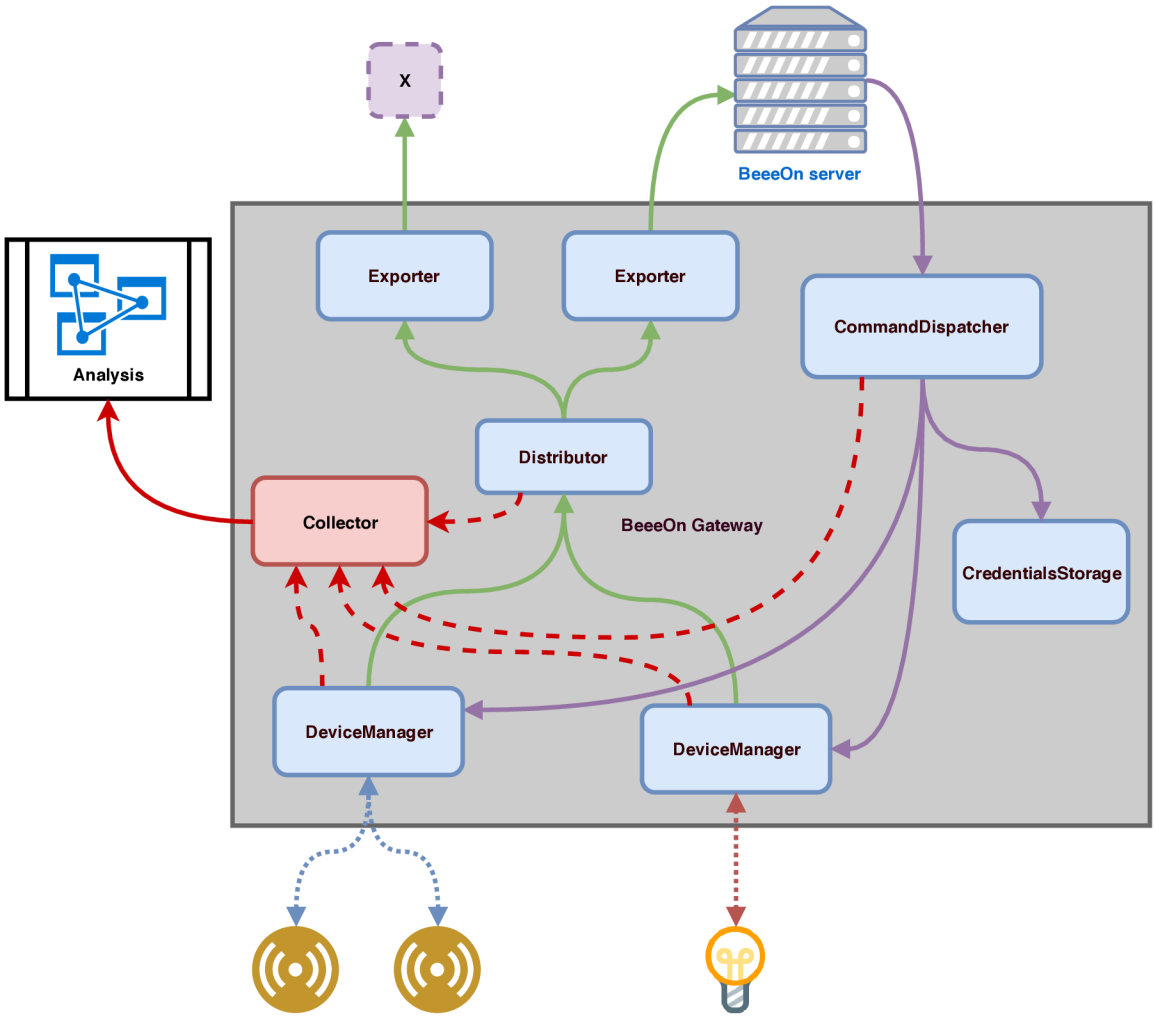

Figure 4.3: Collector in the BeeeOn Gateway

Chapter 5

System for Security Analysis

To reach the goal of this work, which is a secured gateway for wireless IoT protocols, the chosen IoT gateway needs to be secured by a system that will provide analysis of the IoT data flow. This system should be modular and easily extendable with new analysis methods. For these purposes, the Network Measurements Analysis[1] (NEMEA) system seems to be suitable. The information in this chapter regarding the NEMEA system is based on whitepaper published by CESNET [4].

NEMEA is developed as an open-source project, and its maintenance is a responsibility of CESNET Association. It is a modular heterogeneous system for network traffic analysis. Its design respects the stream-wise concept, so the data processing can run continuously in operational memory and does not require any storage space. Moreover, NEMEA can be utilized in many use-cases. It can process live network flows as well as offline traffic traces, it is designed for both operational and experimental use, and it can also be used to analyze other than standard internet networks. The system is very flexible and can be easily extended by new modules which can be implemented in C, C++, or Python language.

A module of the NEMEA system is an application which implements an algorithm or a method that performs a specific task. For example, such a task may be data filtration, anomaly detection, or results reporting. Besides that, each module is built upon NEMEA Framework and uses functionality which is implemented in its shared libraries.

The most important of these libraries is the Traffic Analysis Platform (TRAP) library that implements basic functionality needed by every module, such as TRAP Communication Interfaces (IFC), which allow modules to communicate with each other. An IFC can serve as either an input or an output of a module, and it is an abstraction of several different interprocess communication methods; the most used of them are TCP and UNIX sockets. Thus an input IFC of one module can be connected to an output IFC of another module. Also, every module can have multiple IFC inputs and outputs, and many input IFC can be connected to one output. Data pass through the IFC in the form of short messages (up to 64kB) and create a potentially infinite stream. The specific IFC types along with the parameters specifying where they should be connected, for example, socket name or IP address and port, are passed to a module as command-line parameters, and they are processed by the TRAP library. The key idea is that the main algorithm of the module is entirely abstracted from module communication details, thus a developer can leave all the integration up to the TRAP library.

[1] https://github.com/CESNET/Nemea

Next significant library of the NEMEA Framework is called UniRec and implements a data format used for inter-module communication. Although the IFC also supports another two data formats (unstructured data and JSON), they are seldom used. Inside the NEMEA system, data are transferred typically in UniRec format. It is an efficient binary format for transferring and storage of elementary data records. These records are similar to C structures, but UniRec also supports fields with variable length. However, UniRec records are raw data, and their particular form is given by a template. The UniRec templates can also be defined during runtime, and UniRec fields can be accessed directly without parsing the record. Nevertheless, all UniRec data sent through one IFS has to have the same form given by one specific template. That, however, also radically simplifies data processing, and in most cases, it is not a problem.

The last library that is included in NEMEA Framework is the Common library. This library provides many functions and data structures which are regularly used by network traffic analysis algorithms. Examples include various hash functions, hash tables, Bloom filter, prefix tree, or B+ tree. All the libraries in the NEMEA Framework are written in C language. However, for the first two mentioned, there is a Python API, so it is also possible to use them in python modules. Figure 5.1 shows the structure of a standard NEMEA module built upon NEMEA Framework.

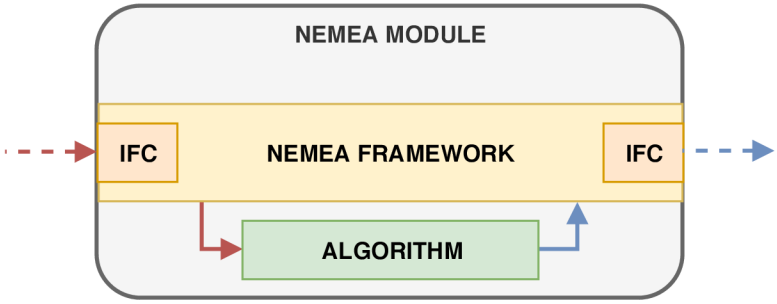

Figure 5.1: NEMEA module built upon NEMEA Framework

Each module is a single program that can be launched from a console. However, an instance of the NEMEA system typically consists of various interconnected modules or even sets of modules. To deploy NEMEA modules with an appropriate interconnection, it is necessary to start its every module with proper parameters. It would be very uncomfortable to do that manually from the console, and it would lead to the opacity of the system and thus cause problems with its maintenance. Therefore, a tool for the NEMEA system control and monitoring was created. This tool's name is NEMEA Supervisor, and it can run whether as a system daemon or in an interactive mode. It uses an XML configuration file which describes a structure of the system in such way, that it defines all demanded modules with their appropriate parameters and module groups, which can be then launched or stopped at once. This configuration file can be changed and reloaded at run-time.

Moreover, the Supervisor also provides a thin NETCONF client that allows a user to change the configuration file via the NETCONF protocol. An instance of NEMEA system is displayed in Figure 5.2. This instance consists of a few interconnected NEMEA modules and two data measurement probes. The figure shows which parts of the example NEMEA instance could be managed by the NEMEA Supervisor.

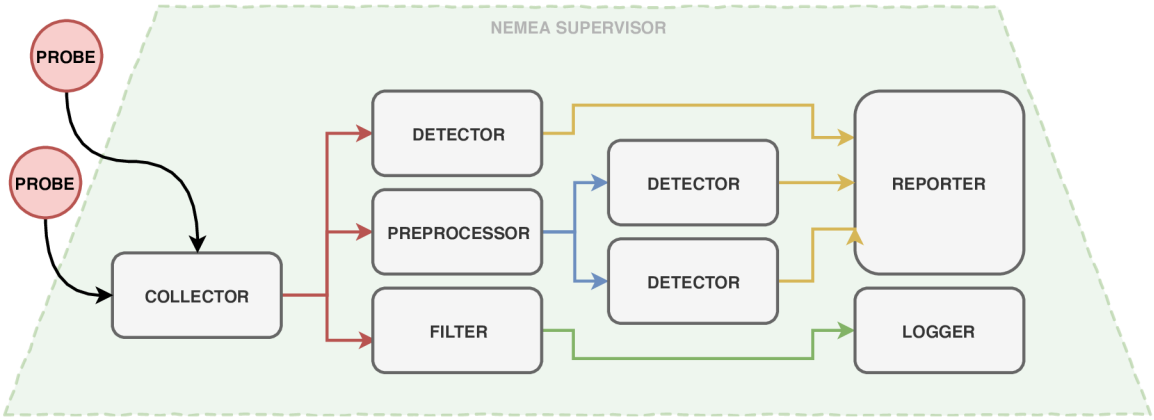

Figure 5.2: NEMEA system with Supervisor

5.1 Data Acquisition

At the beginning of the NEMEA system data processing typically stand modules called collectors. These modules serve for data acquisition and transformation into a NEMEA supported format, typically UniRec. Collectors can use various data sources, for example, internet monitoring probes, Bluetooth HCI interface, or Open Z-Wave Library. A collector can send the data directly to the analyzing modules, or save them into a file for later processing. Four data acquisition modules were developed within the project SIoT [12].

The first module is responsible for acquiring data from from the BeeeOn Gateway. It is called Nemea Collector, and it was developed as the additional BeeeOn Gateway module which, by default, is not compiled nor used, yet this can be changed in Gateway configuration files. It is written in C++ while respecting the implementation practices of the Gateway. The way the Nemea Collector works is that it inherits the BeeeOn class AbstractCollector. The AbstractCollector is a class that serves as an interface through which all the BGW traffic data are accessible. Nemea Collector formats acquired data that are suitable for the NEMEA detectors to UniRec. These data are subsequently sent via UNIX Sockets to Nemea detectors for further processing. Nemea Collector observes and exports data from the following events:

Export - Occurs when data from an end-device are delivered to the gateway. These received data are the object of the observation and exportation.

Dispatch - Occurs when the gateway receives a user command. The object of the observation is the appropriate command.

HciStats - Occurs periodically. The data that are observed are the statistics of a BLE network traffic, which are available on the HCI interface.

DriverStats - Occurs periodically. Observed are the data available on a communication interface of a Z-Wave network.

NodeStats - Occurs periodically. Observed data, which are statistics about Z-Wave nodes within the network, are obtained from the OpenZWave library.

The next SIoT collector is called HCICollector. It is a standalone module which collects all BLE packets with their metadata obtainable on a specified HCI interface. The metadata

consists of the type of a packet, its timestamp, and its direction. Next collector is also a standalone module, which also collects BLE related data. Its name is BLE Advertising Scanner, and its purpose is cyclic listening to BLE advertising channels and reporting all the discovered devices. The last SIoT collector is LoRaCollector which is meant to run on LoRa Gateway[2] developed at VSB Technical University of Ostrava. Its purpose is to format all LoRa messages which the gateway captured into the UniRec format and offer them via IFC for processing.

5.2 Detectors

NEMEA modules which analyze the data flow and identify potential threads are called Detectors. Input IFC of these modules is typically connected to the output IFC of a collector, or some data preprocessor. The detectors then examine received data and search for anomalies or known attack scenarios, and report identified security threads through their output IFCs. Five such modules were developed within the SIoT project [12].

One of them is called WSN Anomaly Detector. It is a universal configurable detector that was developed simultaneously with the Nemea Collector, and these two modules are intended to work as a pair. The principle of WSN Anomaly Detector detections lies in the creation of a network profile and its subsequent comparison with the current network traffic. For these purposes, the detector treats the input data flows as time series and analyzed characteristics are average, moving average, moving variance, and moving median. WSN Anomaly creates a time series for each specific UniRec field specified in a given configuration file. Criteria used for detection which are also particularized in the configuration file, are an unexpected growth, exceeding the determined values, and periodicity violation. However, WSN Anomaly uses one input IFC, while the Nemea Collector can use multiple output IFCs, one for each observed data source. This issue can be overcome by the use of Merger, which is a NEMEA module intended for consolidating many IFC flows into one. After the processing, the WSN Anomaly detector proceeds the information about detected threats via single output IFC.

Next detector serves to detect threats in BLE networks it is intended to work with HCI Collector mentioned above. Its name is BLE Pairing Detector, and its purpose is to identify unexpected BLE pairing procedures. Many BLE security threats stem on the interception of the pairing procedure. Therefore, unexpected pairing procedures can indicate an attack, where an attacker forces already paired devices to go over the pairing procedure again.

The following group of SIoT detectors that consists of LoRa Airtime Detector, LoRa Distance Detector, and LoRa Replay Detector, is developed to increase the security of the LoRaWAN network. All these detectors are designed to work with the LoRaCollector, and thus they expect their input to be LoRa packets in UniRec format. The purpose of the Airtime Detector is airtime regulation for individual LoRaWAN end-devices. LoRaWAN protocol specifies the maximum on-air time for each device, and this detector is intended to detect its exceedance. LoRa Replay Detector detects potential attacks that can cause the DoS of a LoRa end-devices. These attacks misuse the vulnerability connected to LoRa devices which use APB joining procedure, by injecting previously captured messages at the right time. The last LoRa detector detects physical movement of LoRaWAN end-devices. The detector considers that a device had been moved when the Received Signal Strength Indicator (RSSI) in the newly received packet and the previously stored RSSI for the device

[2] https://lora.vsb.cz/index.php/building-rpi-gateway/

differs. However, this method is not highly accurate, and it is not possible to compute the exact device position. Nevertheless, it also does not require sophisticated algorithms, nor much computing power, and the information that the device had changed its position can be precious.

Chapter 6

Gateway design

The goal of this work is to create a secured IoT gateway, which will consist of BeeeOn Gateway and existing detection systems and run on Turris Omnia. The resulting solution must meet several functional and non-functional requirements to fulfill this goal and also to be the quality. Functional requirements specify the resulting solution functionality itself. One of these essential functional requirements is to preserve the functionality of the BeeeOn Gateway, which can communicate with and control IoT devices. In order to secure this communication, detection systems will always be running along with the gateway. All these systems will run on the router. Routers are typically restarted once a time and have to work without any user interaction for a long period. Therefore, the system must be started automatically with the router and run continuously. Another main requirement is system interaction with the user. As long as the detection systems and nor the BeeeOn Gateway cannot react in the way to prevent or stop a potential attack, in the case of a detected security threat, the system will only warn the user. Also, the resulting system has to be configurable, meaning that it will be possible to choose detection systems and configure them, as well as BGW.

Regarding the limiting properties of the system, i.e., non-functional requirements, two of them result from the nature of Turris Omnia. Although it is one of the more powerful routers, it should be taken into account that it has considerably less computing power than a regular PC. Therefore it is necessary for the resulting system to have relatively small demands on computing power. It is also noteworthy that this router uses the TurrisOS operating system based on OpenWRT, and thus, the resulting system will be targeted to run on and also tested on this platform. Another non-functional requirement is expandability. It is likely that new detection systems will emerge and that the existing ones will somehow change or improve. Hence the architecture of the resulting system will be proposed the way that will ensure its easy extendibility and independence of its subsystems.

6.1 System proposal

The proposal of the resulting secured IoT gateway system is shown in figure 6.1.

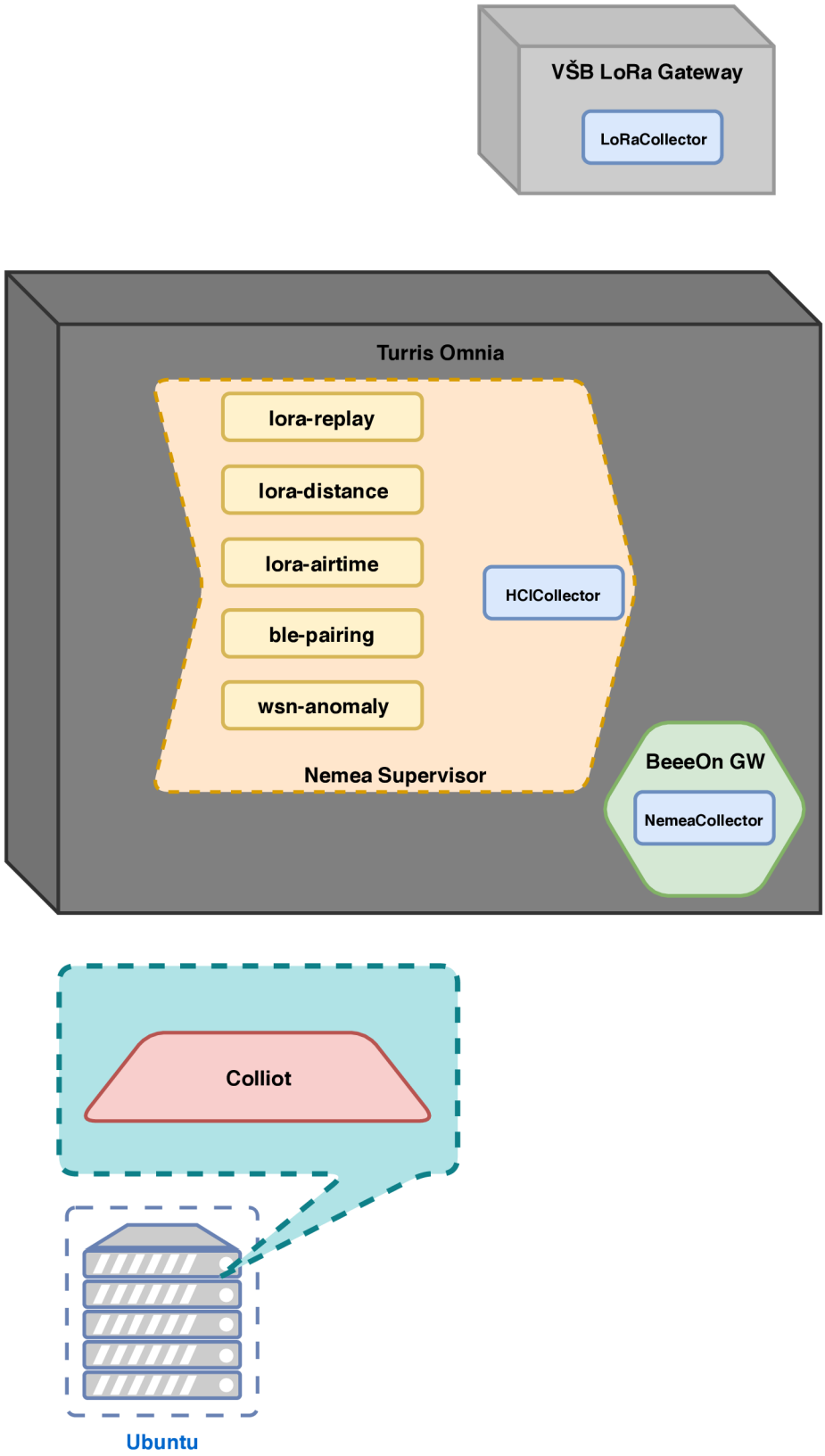

Figure 6.1: Proposal of the secured IoT gateway system

The design consists of two hardware components, one virtual machine, and ten software components. The idea is the BeeeOn Gateway serves as IoT gateway. Thus the BGW connect to and control IoT end-devices and also communicate with the server to send data and receive commands. However, the BGW is extended with one additional module. It is the Nemea Collector [16] which, along with the HCI Collector and the LoRa Collector, serve as a data source for the detection systems. Figure 6.2 displays the usage of BGW in the resulting system.

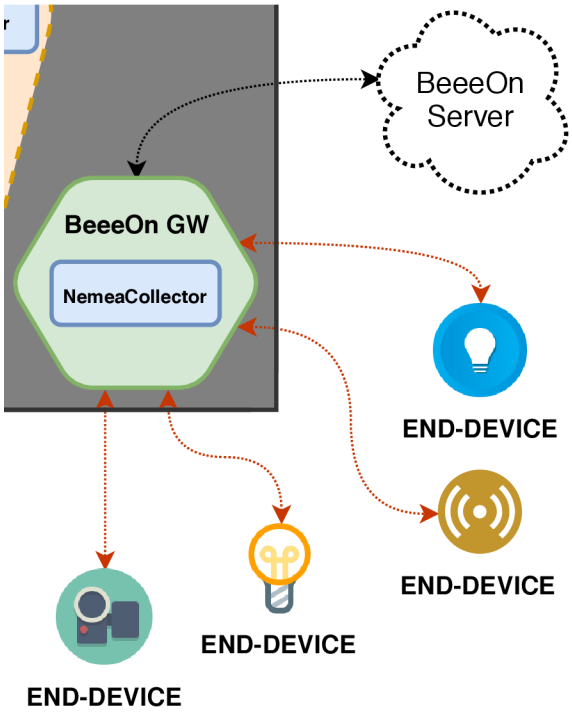

Figure 6.2: BeeeOn Gateway as a part of the secured IoT gateway system

The proposal shows the BGW secured using the detection systems, that were developed within the SIoT project and are built on the NEMEA framework. These detection systems are:

WSN Anomaly Detector - This detector will retrieve data from the Nemea Collector and analyze them to detect potential DoS attacks in the Z-Wave network.

BLE Pairing Detector - This detector will retrieve data from the HCI Collector and analyze them to detect suspicious or unexpected BLE pairing process.

LoRa Airtime Detector - This detector will retrieve data from the LoRa Collector and analyze them to detect eventual exceeding the allowed transmission time limit of each LoRaWAN end-device.

LoRa Distance Detector - This detector will retrieve data from the LoRa Collector and analyze them to detect changes in LoRaWAN end-devices position.

LoRa Replay Detector - This detector will retrieve data from the LoRa Collector and analyze them to detect potential DoS attacks in the LoRaWAN network.

A central hardware unit of the system, where also the most crucial software parts of the system are running, is the Turris Omnia router. All the detectors and also the HCI Collector are NEMEA Modules designed as stand-alone programs. As long as it would be complicated to run and manage all these modules individually, it is reasonable to use a program that encapsulates them and ensures their unified management. Therefore the proposal shows NEMEA Supervisor that manages all stand-alone NEMEA modules on the router. The usage of NEMEA Supervisor will make it easy to enable, disable, or anyhow configure these modules and also to automatically start them along with the operating system because the Supervisor can work as a system daemon.

Nevertheless, three above mentioned detectors detect threats in LoRaWAN networks, and LoRa Collector is designed to be their data source. This Collector was designed and developed to run on the VSB LoRa Gateway, which is, therefore, the second hardware part of the resulting system. In the resulting system LoRa Gateway, however, does not fulfill any other purpose except being a platform for LoRa Collector. Figure 6.3 shows data flows between collectors and detectors in the resulting system. Specifically, it displays which collectors serve as the data source to individual detectors.

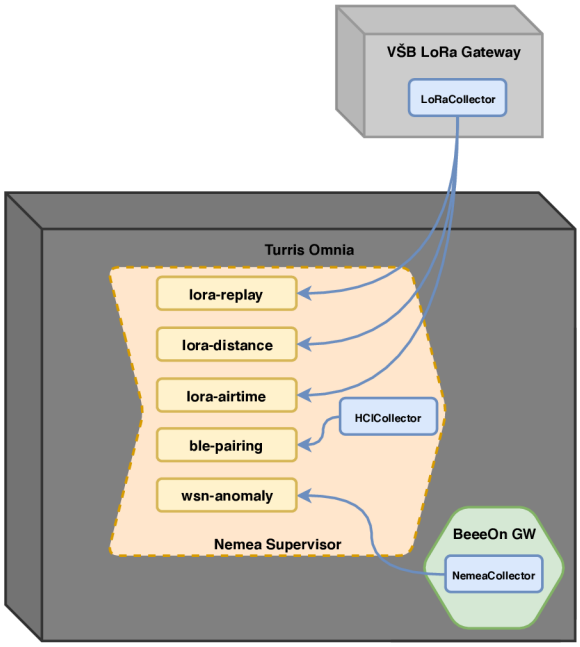

Figure 6.3: Collectors as data sources for detectors

The last software part of the system is the Coliot IoT collector. The Coliot role in the system is to collect data from the detectors to store them permanently and present them to the user. The design shows Coliot running on a Ubuntu virtual machine. The Coliot system was developed to run on Ubuntu and Debian, and no installation script nor a package for other operating systems is available at the time of writing. The idea of user interaction with the security system is displayed in Figure 6.4. The figure shows that all the detectors send data from the router to the Coliot system running on a virtual Ubuntu machine. A user can interact with the system by connecting to the Coliot system.

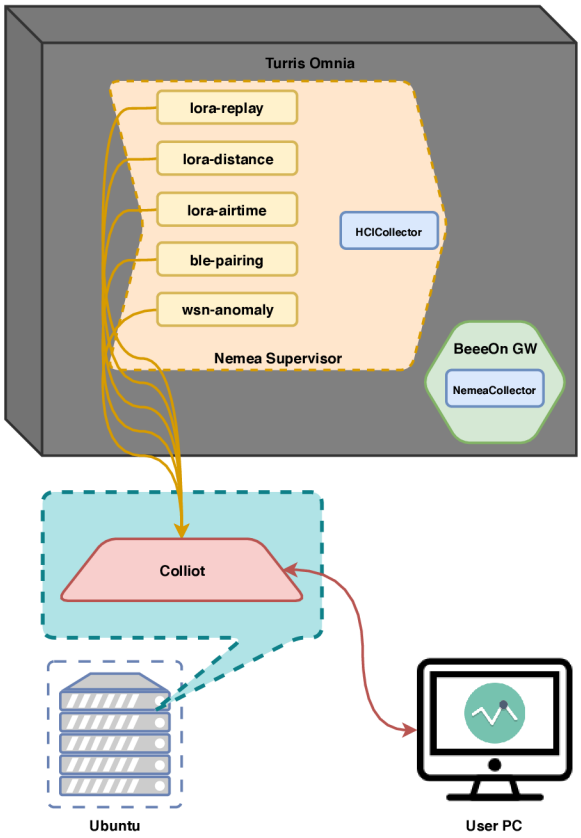

Figure 6.4: Role of the Coliot Collector in resulting system

Chapter 7

System integration

It is necessary to integrate and interconnect all the mentioned components to build a resulting system as designed in the previous chapter. Therefore following activities had to be done:

Creation of the integration tests: It is a good practice to determine how will the resulting system work at the beginning of the development phase. That means, in particular, the creation of some way to verify correct system functionality. The resulting system will be tested by a testing script using pre-set data sets. Therefore it was necessary to design and implement this script.

Nevertheless, the best way would be to test the resulting system on real IoT network with real attacks. However, the creation of a testing environment meeting these requirements is not possible within this work. It would require many IoT devices and also other hardware components suitable for attacking.

Integration of the Nemea Collector to the BeeeOn Gateway: The Nemea Collector was developed directly for BeeeOn Gateway but has never been incorporated into the official repository. However, to use BGW in the secured IoT gateway system for the long run, it is necessary to incorporate it to reflect any changes in BGW and thus avoid incompatibility over time. Moreover, the incorporation facilitated the creation of packages as described in the next indent, because the BGW can be compiled directly from official source codes available on Github.

Creation of packages for TurrisOS: A significant part of the system run on the Turris Omnia router with OpenWRT-based TurrisOS operation system. Therefore it was necessary to create packages of the required programs that are suitable to be installed and running on the router. Specifically, these programs are BeeeOn Gateway, NEMEA Supervisor, HCI Collector, WSN Anomaly Detector, BLE Pairing Detector, LoRa Airtime Detector, LoRa Distance Detector, and LoRa Replay Detector.

Instalation and configuration of the created packages to the Turris Omnia: After their creation, the packages need to be installed and tested on the router. Then, to use them as one secured system, they have to be suitably configured. Such configuration activities included, in particular, creating a configuration for the NEMEA Supervisor and edit BGW configuration files. With the proper configuration, the Supervisor arranges correct interconnection of all the NEMEA modules. Correctly configured

BGW preserves its full functionality on the Turris Omnia and interconnects with appropriate NEMEA modules.

Creation of the virtual disk with configured Coliot system: Another needed functionality of the resulting system is interaction with the user, which is provided by the Coliot system running on a virtual Ubuntu machine. Consequently, such a virtual machine needed to be created, and the Coliot system needed to be installed to it. The Coliot system, moreover, had to be then appropriately configured to be able to correctly recognize incoming data and visualize it to the user in a useful way.

Inter-hardware interconnection: A necessary step was also connecting all hardware components. That included the Turris Omnia, VSB LoRa Gateway and also the machine that the virtual Ubuntu will run on. However, it was not only needed to interconnect them with an ethernet cable, but also to connect appropriate programs that need to be connected but run on different devices. These programs are, specifically, LoRa Collector which runs on VSB LoRa Gateway and is meant to serve as a data source for all LoRa detectors that run on the Turris Omnia. , and also the Coliot Collector, which will run on a virtual machine, but its purpose is to collect data from all detectors.

7.1 Creation of tests

First of the activities is the creation of a testing script that will provide integration tests of the system. Figure 7.1 shows the proposed, and also implemented, principle of its operation.

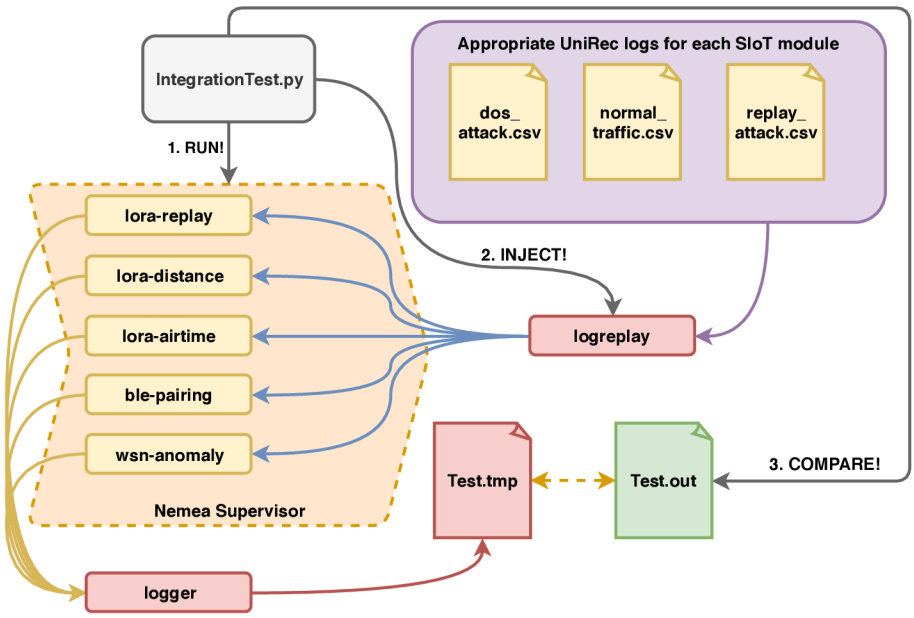

Figure 7.1: Integration test script process

The way, how it works, is that at first, it launches an instance of the NEMEA system that consists of all detectors developed within the SIoT project. Then it injects appropriate data sets containing captured network traffic to each detector and, finally, it compares their outputs with the pre-created expected outputs.

All the SIoT detectors are published and maintained in a central git repository on Github called NEMEA-SIoT. However, none of these detectors had an automated tool to test it and, furthermore, the majority of the detectors did not even include example datasets suitable to demonstrate their functionality. These facts have proven to be a problem in creating an integration testing script and motivated the creation of automated tests for the modules within NEMEA-SIoT repository. The figure 7.2 displays the principle of proposed, and also implemented automated tests.

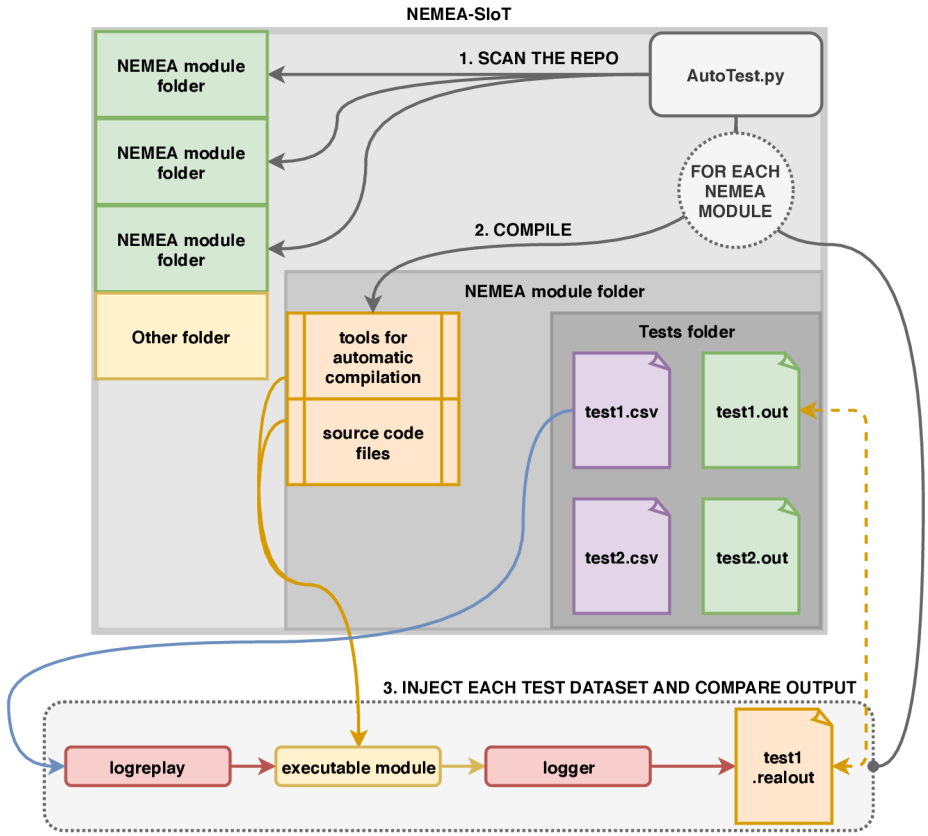

Figure 7.2: Auto test script for NEMEA-SIoT repository

The idea is that a single script located at the uppermost level of the repository provides the testing. This script tests each module separately and thus provides unit tests of NEMEA-SIoT systems. Firstly it scans the repository and identifies all folders containing NEMEA module written in C language and also determines, wheater the module includes tests. Then the script tries to compile all modules and verifies the result of this compilation. The modules which were successfully compiled, and include tests, are then tested. For each module, the testing process begins with the identification of all test datasets and appropriate files with expected module output after their processing, and then goes as follows:

1. The script launches the module which is currently under testing, waits for one second and ensures that the module is correctly running.

2. The next phase needs two additional NEMEA modules are - Logreplay and Logger. The first test data set is injected to the module input IFC using the NEMEA Logreplay, and the module output is captured by the NEMEA Logger.

3. After a module process the data, the script shuts down all launched programs. The end of data proceeding is determined the way that the testing script waits until the tested module shuts down. This shut down is caused by the fact, that after replaying the complete test data set, NEMEA Logreplay sends a special EOF UniRec record. As soon as the tested module receives this EOF record, it ceases its activity.

4. Finally, using the shell diff utility, the script compares captured output with the expected one and evaluates the result.

5. And then the script repeats all steps for the next test dataset.

However, it takes quite a lot of time to test the whole repository, as there is a timeout for every single test, as results from the first step mentioned above. Moreover, some of the test datasets are huge, and thus, their processing takes a few seconds. Furthermore, the compilation takes much time too and is not always necessary. Therefore, the script supports arguments by which a user can specify the modules that are demanded to be tested, and also can turn off the compilation or testing. Also, the paths to NEMEA Logger and NEMEA Logreplay can be specified by script arguments. Thus it not inevitable to have them installed.

For the script to work properly, it is necessary that modules have a uniform and clearly defined structure. Nevertheless, the structure of individual modules differed. Therefore, the following rules have been agreed upon and implemented into the repository:

- Every module is stored in the directory with a name created as the lowercase name of the module with dashes instead of spaces (WSN Anomaly → wsn-anomaly).

- Module source codes and files needed for automatic compilation is stored in the uppermost level of a module directory.

- The compilation of a module creates in an executable file with the name identical to the repository but with the siot- prefix (wsn-anomaly → siot-wsn-anomaly).

- In each module directory, a directory named tests is present. This directory serves as a storage place for file pairs composed of a file containing a test dataset and a file containing modules expected output after the dataset processing. Both the files, forming the pair, have an identical name, but the suffix of the dataset file is .csv, and the suffix of the expected output file is .out.

- A module directory can also contain a special file named tests.json. In this file, two sets of shell commands and a set of arguments are specified. One set of the shell commands is executed before the module is launched. Then the module is launched with the specified arguments, and finally, the second set of shell commands is executed. By this, it is possible to overcome the fact that besides the common NEMEA Trap arguments, a module could support or even require other specific arguments.

This repository modification and auto test script creation significantly facilitated the development of the integration test script, which can also use the mentioned test datasets.

The work of the integration test script and auto test script is significantly alike. The main difference is that the integration test launches all modules at once. For this purpose, the NEMEA Supervisor is used. Also, the integration test does not have anything to do with the module compiling; it tests installed modules.

However, this test does not cover any other part of the resulting system than the detectors, which are running on Turris Omnia. Nevertheless, the creation of an automated test that will cover a more significant part of the resulting system will be overcomplicated. Besides, this test covers the heart of the securement - co-work of all used detectors. Moreover, other parts of the system could be quite merely tested manually.

7.2 Nemea Collector incorporation

Another task was to incorporate Nemea Collector into the BeeeOn Gateway repository. Thereby it is ensured that the potential future changes in the BGW will reflect on the collector and thus it prevents incompatibility from occurring over time. This reason proved to be adequate as the original Nemea Collector was even no longer compatible with the latest version of the BGW. The reason for this incompatibility was that names of some BGW methods had changed. Hence it was necessary to reflect these changes in the Nemea Collector source codes to make it possible to compile the BGW with it.

Furthermore, the previous use of the Nemea Collector was not in line with the principles that are followed when developing the BGW. Therefore, the Nemea Collector source codes and their location within the BGW repository have been changed. Changes also occurred in the BGW configuration files and files needed for its compilation. These modifications have introduced Nemea Collector as one of the optional BGW modules that can be easily switched on and off in the BGW configuration. In this state, the Nemea Collector has been incorporated into the official BGW repository, and by default, it is turned off.

7.3 TurrisOS packages

Packages for the TurrisOS are necessary to install and run secured IoT gateway system on the Turris Omnia. In addition to packages of the already mentioned programs, from which the resulting system will consist, it was also necessary to create packages that are not included in the TurrisOS package system, but the resulting system is dependent on them, or are needed to test or link some parts of the system. Specifically, except the SIoT modules and BeeeOn Gateway, demanded packages are:

- **GLib2 and Poco libraries**
  The glib2 package and also a package containing the Poco libraries are needed as the BGW depends on them, but the TurrisOS package system does not include them.
- **NEMEA Framework**
  All the NEMEA modules, which include all SIoT detectors and collectors, are dependent on the NEMEA Framework. Consequently, its package is required.
- **NEMEA Logger and Logreplay**
  These two NEMEA modules, and, therefore, their packages, are required for resulting system testing.

- **NEMEA Merger**
  Because of the fact, that the Nemea Collector outputs data through as many IFCs as many events it listens to, the NEMEA Merger is required as the interface between the Nemea Collector and the WSN Anomaly Detector which uses single input IFC.

Furthermore, a new NEMEA module was created due to the fact that the NEMEA instance in the resulting solution has to run on the router with TurrisOS. It is a lightweight version of the NEMEA Supervisor called NEMEA Supervisor Lightweight (SupervisorL). Among its most significant benefits compared to the classic NEMEA Supervisor belongs the fact, that except the NEMEA Framework it has no dependencies besides the commonly available TurrisOS packages. Moreover, it is much less sophisticated and demands significantly less computing power than the regular Supervisor. However, the essential functionality for the secured IoT gateway system remains. The NEMEA SupervisorL can start and stop all demanded NEMEA Modules with their interconnections at once. For its work, SupervisorL needs a single configuration file where the required NEMEA instance is specified.

For the reason that the SIoT project also aims at the possibility of the deployment of developed modules on commonly available network elements, especially routers, there already has been an initiative to create SIoT NEMEA modules packages. Moreover, as long as the widely spread, and for this purposes most suitable operating system used on such devices is Linux-based OpenWRT, this initiative is mainly aimed at OpenWRT packages.

Consequently, some packages for OpenWRT and also for the TurrisOS has already existed. Nevertheless, few essential packages for the secured IoT gateway system were missing. Moreover, a principle of releasing, maintaining, versioning, and testing these packages has not been established. The individual packages were in different locations, and they did not come from the latest module versions. Furthermore, they were not tested and often contained errors that made them unusable.

Therefore it was necessary to establish a uniform system for package releasing, testing, and maintaining. For this purpose, a branch in the NEMEA-SIoT Github repository was created. A name of this branch is turris-secured-gateway, and its structure is designed to store and manage the packages required to create the secured IoT gateway system on the TurrisOS. Along with the appropriate packages, it also contains text files with information related to them and instructions needed for the system installation. The structure of this branch is illustrated in figure 7.3 and is as follows:

Uppermost level: Repository root on this branch contains directories that serve to store packages. Each directory stores packages of one part of the system. In these parts, packages are grouped based on which project they are related. Therefore there are three of this directories - one with the name siot-modules for NEMEA modules developed within the SIoT project, the other with the name nemea for all other NEMEA related packages and the last one, named beeeon-gateway, for all packages necessary for the BGW installation.

Also, there is an installation script and a readme file in the root directory. The installation script serves to install whole secured IoT gateway system. That means it installs all packages in the proper order, as shown in Figure 7.4. For this purposes, it uses installation scripts in subdirectories, which are described in the following indent. Firstly it calls installation script in the Nemea directory. It is because the SIoT NEMEA modules and also the BeeeOn Gateway with enabled Nemea Collector are

dependant on the NEMEA Framework. The order of installing siot-modules and beeeon-gateway does not matter, yet the script calls the siot-modules installation script as the second and the beeeon-gateway installation script as the last.

The readme file present in the repository root contains information about the repository itself, its purpose and the guide on how to install the whole secured IoT gateway system using the installation script.

Each package group directory: The idea is that packages related to one project are kept within one repository subdirectory. However, the beeeon-gateway directory also includes glib2 and poco-all packages, as the BGW depends on it. Moreover, the BGW, as well as NEMEA SIoT modules, is also dependant on the NEMEA Framework which package is stored in the nemea directory. Therefore, these directories cannot be regarded as independent package groups nor as package groups of programs that have been developed within a single project. Still, this division best reflects the relation of packages to individual projects and brings benefits to their release, maintenance, and retention of associated information.

That information is held in a readme file which is present in each of these subdirectories. Each of these readme files contains a description of the packages within the appropriate directory, their dependencies, their installation procedure, and also a TODO list linked to them. Examples of items that TODO lists are meant for include bugs found in the latest version or proposed enhancements for the future. The last thing present in package group directories is installation scripts. These scripts contain shell commands which reflect installation guide in the relevant readme file and thus automize the installation.

For this work, the packages were created from the latest available versions at the time of writing:

- BLE Pairing Detector - commit c083deb in branch NEMEA-SIoT/master[1]
- WSN Anomaly Detector - commit 145edd4 in branch NEMEA-SIoT/master[1]
- LoRa Airtime Detector - commit d7a3730 in branch NEMEA-SIoT/master[1]
- LoRa Distance Detector - commit 2049474 in branch NEMEA-SIoT/master[1]
- LoRa Replay Detector - commit 2049474 in branch NEMEA-SIoT/master[1]
- HCI Collector - commit 202fa5c in branch NEMEA-SIoT/master[2]
- NEMEA Merger - commit 9a3c031 in branch Nemea-Modules/master[2]
- NEMEA Logger - commit 90d9643 in branch Nemea-Modules/master[2]
- NEMEA Logreplay - commit dbf6d8d in branch Nemea-Modules/master[2]
- NEMEA SupervisorL - not publicly available
- BeeeOn Gateway - commit 121ee07 in branch gateway/master[3]

[1] https://github.com/CESNET/NEMEA-SIoT/tree/master/
[2] https://github.com/CESNET/Nemea-Modules/tree/master/
[3] https://github.com/BeeeOn/gateway/tree/master/

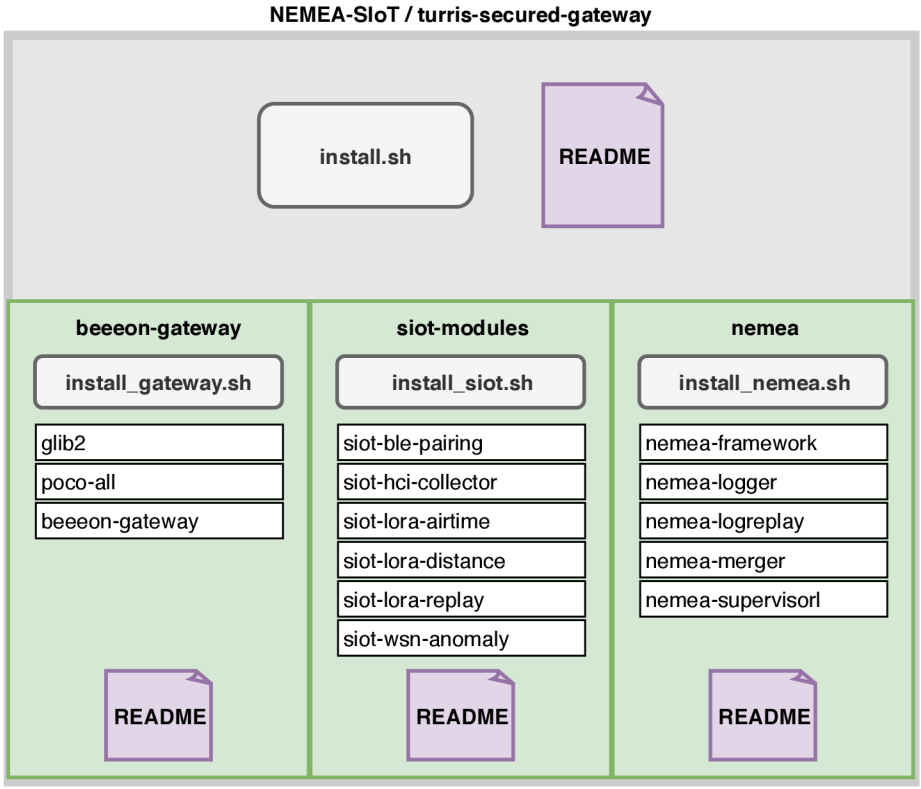

Figure 7.3: NEMEA-SIoT repository - branch turris-secured-gateway

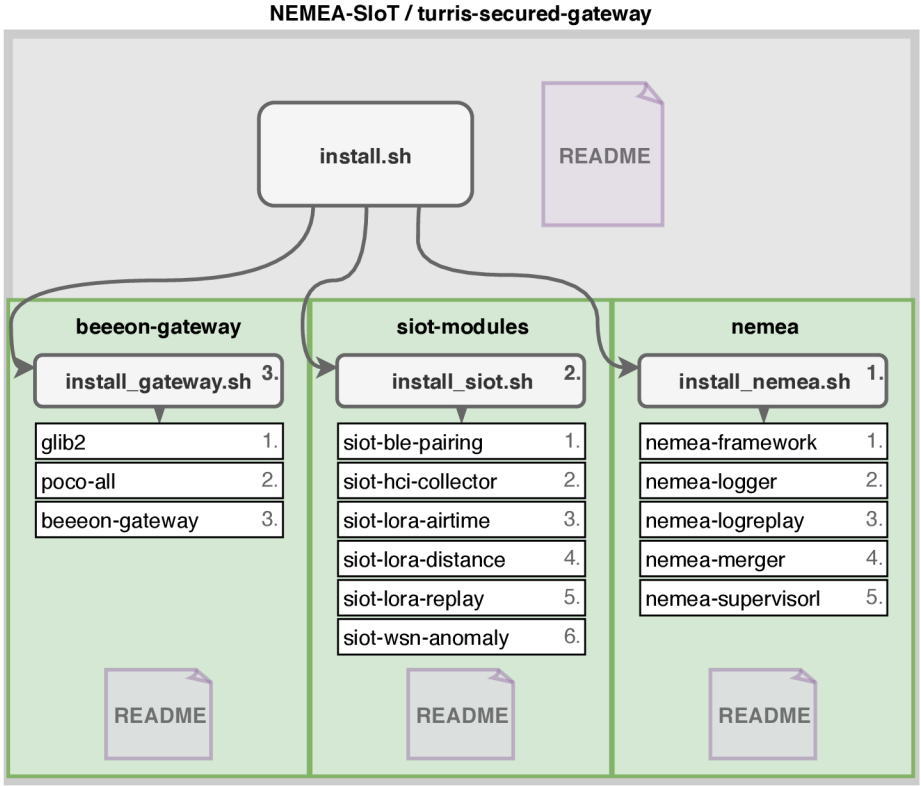

Figure 7.4: TurrisOS package installation order

7.4 Coliot

To provide relevant data to a user, it was necessary to set up the Coliot system. This system is designed to work with the UniRec data. These data are acquired by Coliot Collector and stored in a SQL database. The Coliot Collector is a python NEMEA module, which uses a single input IFC. For its proper work, it needs a set of templates which contain two sections – MAIN and FIELDS. The FIELDS section stores definitions of UniRec fields in the format `NAME_OF_FIELD = datatype`. When the Coliot Collector receives first UniRec record, it goes through all templates to find the one which fields definitions match with the fields of the received record. Then it stores the received data in the database to a table which name is given in the MAIN section of the appropriate template. Structure of this table is generated based on the FIELDS definitions. Therefore, to correctly store all the detection results in the database, it was necessary to create a Coliot Collector template with appropriate FIELDS section for all the detectors.

As a GUI, the Coliot system uses Apache Superset[4] web application. The way it works is that it has access to the database and allows user creating dashboards with graphs and tables from the available data. To make the detection results user-friendly to read a dashboard that shows all the significant detection results in one place was created.

7.5 System deployment

The most significant part of the system is BeeeOn Gateway and the SIoT NEMEA instance on the Turris Omnia. To deploy these systems, it is necessary to install created packages to the router. That was done using before mentioned installation script available in the NEMEA-SIoT repository on the branch turris-secured-gateway.

However, BGW supports WiFi, Z-Wave, BLE, and IQRF protocol, but the router only provides a communication interface for WiFi. Therefore it was necessary to plug additional modules (dongles) into the router providing communication interfaces for Z-Wave, BLE, and IQRF protocol to conserve full functionality of the BGW. Moreover, the HCI Collector also draws data from the HCI interface and thus a Bluetooth dongle. Figure 7.5 shows the Turris Omnia router with all additional required dongles and their connection to the secured IoT gateway system parts. Also worth mentioning is that the Turris Omnia has only two USB input interfaces, and since three dongles are needed to use the full system functionality, it was unavoidable to use a USB hub.

[4] https://superset.incubator.apache.org/

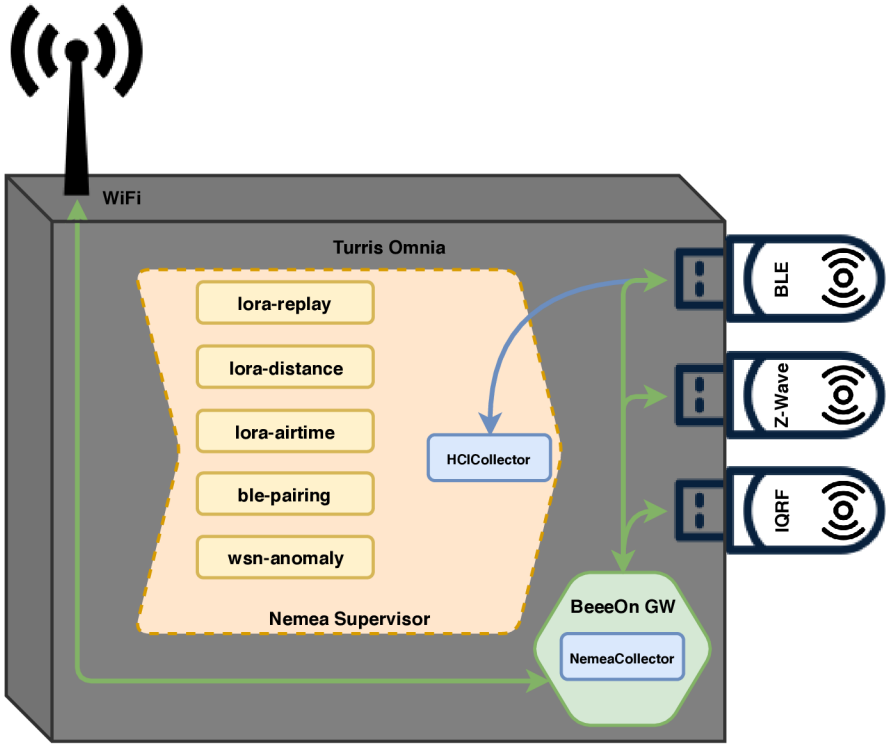

Figure 7.5: All necessary dongles plugged into Turris Omnia

For this work, Turris Omnia router with clean OpenWrt omnia 15.05 system that has disabled auto-updates is used. This system, by default, do not recognize the abstract control model (ACM) devices, and the used Z-Wave dongle represents itself an ACM device. Therefore it was necessary to install the kmod-usb-acm package to it.

In addition to preserving the full functionality of BGW, it was also necessary to properly interconnect the parts of the system running on the router. Specifically, the HCI Collector with BLE Pairing Detector and Nemea Collector with WSN Anomaly Detector. All NEMEA modules use IFCs as an input and output interface. Specification for those IFCs is given to stand-alone modules as arguments at their launch. Consequently, interconnections of these modules can be handled by the NEMEA SupervisorL, as it is possible to specify arbitrary arguments for each NEMEA module in its configuration file.

However, the Nemea Collector is not a stand-alone module, but its a part of BGW. Thus it cannot be configured in the SupervisorL configuration file, but its output IFCs are specified within the BGW configuration. Moreover, in the resulting solution, the Nemea Collector uses multiple output IFCs. Specifically, five of them – each for one observed BGW event. On the other hand, the WSN Anomaly Detector uses a single input IFC. To overcome these issues, the NEMEA Merger was used as an interface between WSN Anomaly and Nemea Collector. As long as NEMEA Merger is a standard stand-alone NEMEA module, it can also be managed by the NEMEA SupervisorL.

Besides the connections within the router, all the detectors must be connected to a Coliot Collector which runs on a virtual machine. Also, the LoRa detectors must be connected to the LoRa Collector running on the VSB LoRa Gateway. Nevertheless, all the mentioned programs are NEMEA modules. Therefore their interconnection can be handled by the use of TCP IFCs. That means that on all detectors on the Turris Omnia must open an

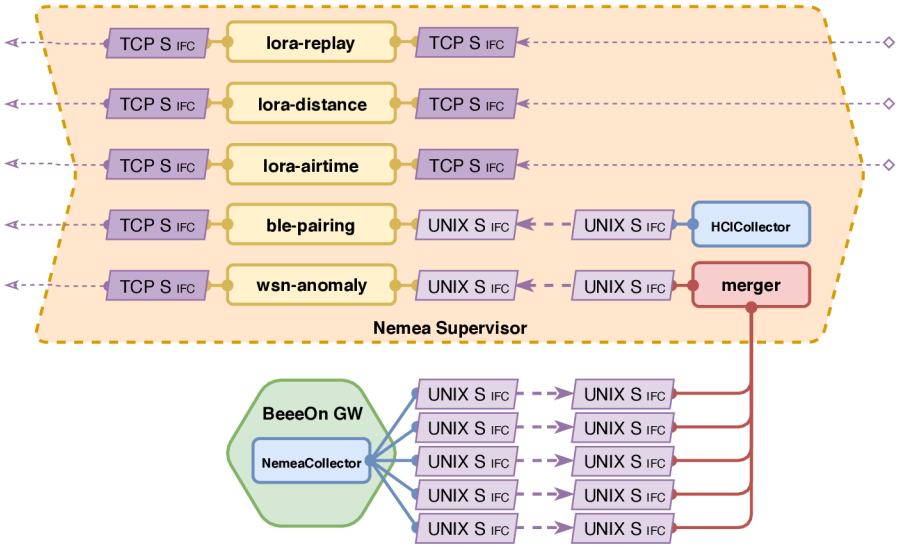

Figure 7.6: IFCs of NEMEA modules running on the Turris Omnia

output TCP IFC to which the Coliot system will connect. Besides, LoRa detectors have to use input TCP IFC to connect to the LoRa Collector. However, the type of IFC does not affect the way of its specification. Hence, these TCP IFCs can also be specified in the SupervisorL configuration file.

Therefore the creation of appropriate configuration file for SupervisorL entirely solved connection requirements of the system parts running on the Turris Omnia. Figure 7.6 displays all system parts running on the router and all the IFCs they use. Arrows in the figure between the IFCs always point from an output IFC to an input one and thus indicate the direction of the data transfer.

Nevertheless, it was also necessary to set up appropriate connections on the other two devices that are part of the resulting system. The Coliot system running on the virtual Ubuntu uses a python script called Coliot Collector for the data acquisition. This script is built upon NEMEA Framework and uses the IFC as an input interface. However, the Coliot Collector is designed to use a single input IFC, while Coliot system needs to be connected to five detectors. Therefore, five instances of Coliot Collector were used to interconnect the Coliot system with all the detectors running on the Turris Omnia. Each of these instances connects with its input TCP IFC to the output TCP IFC of one detector.

The LoRa collector which runs on the VSB LoRa Gateway and serves as a data source for LoRa detectors is also a NEMEA module. To connect it with the detectors it was necessary to launch it with a correctly specified output TCP IFC. Subsequently, all the LoRa detectors connected to that IFC, as long as it is possible for multiple input IFCs to connect to a single output one.

Chapter 8

Results

The resulting functional sample of secured gateway for wireless IoT protocols was created by integrating many systems that are being developed by multiple developers across multiple projects. Also, its overall functionality depends on the functionality of the used systems. The fact that most of these systems are still under development, and they have often not been systematically tested has proven to be critical. Moreover, the development of these systems did not take into account that they would be used as a single unit. Consequently, several problems had to be solved during their integration so that the resulting system could be created.

The first problem solved was that the functionality of the individual detection modules could not be uniformly verified. To overcome this deficiency, a script for automated testing of modules developed within the NEMEA-SIoT repository was implemented. This test revealed many shortcomings. The main hindrance was that no data was available to demonstrate the functionality of any testable module except WSN Anomaly Detector. Furthermore, despite the existence of data samples for the WSN Anomaly Detector, the expected module output after their processing was not defined. Therefore, the correct functionality of any SIoT module could not be verified at all. Another revealed deficiencies included inappropriate access rights on one module's script needed for its automated compilement, improper names of resulting executable files after modules compiling and also shortcomings in the structure of module s directories. An overview of the problems found by the auto test is shown in Table 8.1.

The test was incorporated into the NEMEA-SIoT repository, and modules developers were made aware of their modules shortcomings. All the revealed deficiencies were fixed, and the data suitable for modules testing along with the expected outputs after their processing were added. However, this exposed a deficiency in the NEMEA Logger, which is used by the testing script to capture modules outputs. The problem was the output format of MAC Address datatype which can occur in the UniRec record. The NEMEA Logger formatted this output with the usage of a library which behavior differs depending on the operational system. Consequently, the tests were passing on some computers yet failed on others. As long as the use of Logger is the only suitable way of capturing modules output, its developers were notified, and this issue has been fixed.

Module	Structure	Auto compilement	Executable name	Test data
BLE Pairing	✓	✓	×	×
WSN Anomaly	×	✓	×	✓
LoRa Airtime	✓	×	×	×
LoRa Distance	✓	✓	×	×
LoRa Replay	✓	✓	×	×

Table 8.1: Module deficiencies revealed by the auto test script

As soon as all modules were unified and tested, the deployment on Turris Omnia followed. Packages of all necessary modules were created using the existing SIoT environment designed for these purposes. Also, an integration test script suitable to test the system on the router was implemented. However, after the packages were installed, the test did not pass, and it has revealed various package-related problems. Multiple modules were crashing after they received data via the NEMEA Logreplay. The detectors LoRa Distance and WSN Anomaly were falling on a bus error, and LoRa Airtime was crashing on a segmentation fault. Besides, the LoRa Replay detector output was different than expected. Moreover, the NEMEA Logreplay installed from the package was not able to parse timestamps present in the BLE Pairing testing data sets. Also, experimental testing of the whole system shown, that Coliot system was not ready to process UniRec records containing MAC addresses.

These issues have been reported, and most of them were successfully fixed. Table 8.2 summarizes the problems that occurred during testing of the resulting system and whether they were solved. The errors connected to the LoRa Distance and WSN Anomaly persists. Therefore, these detectors are not suitable to be used in potential production as a part of the resulting system.

Problem	Solved
NEMEA Logreplay - unable to parse timestamps	✓
WSN Anomaly - bus error	×
LoRa Airtime - segmentation fault	✓
LoRa Distance - bus error	×
LoRa Replay - inappropriate output	✓
Coliot system - unable to process records with MAC Address	✓

Table 8.2: Issues associated with installed packages

Nevertheless, all the other parts of the system work as expected except a small problem that occurred after the LoRa Airtime segmentation fault fix. About 1 out of 250 its UniRec output records is improper. However, this is not a crucial issue, the detector can be used, and the system can be deployed in its enlighted version. The connection between VSB Lora Gateway, Turris Omnia router, and Coliot system running on Ubuntu VM functions properly, and a user can display detection results using Coliot web interface. This functionality was experimentally tested by running the entire system, sequential injection of test data sets to working detectors and monitoring results via Coliot system. Experimental testing also included testing of the BeeeOn Gateway. Its functionality was tested via the Testing Center, which is one of its modules, allowing its control via a console interface. Everything worked as expected. The Coliot system shown appropriate detection results, and the BGW was able to connect to and control IoT end-devices.

The detectors running on Turris Omnia were also subjected to stress tests. The test scenario was that the modules were launched and test data sets were injected to their inputs in an endless cycle. During the test, system resource consumption was monitored using the shell top utility. None of the functional detectors loaded the processor to even one percent, nor consumed at least two percent of virtual memory (VMS). The same results of consumed resources were observed during the BGW experimental testing. Therefore, it can be stated that the router is capable of running the system without any problems.

Chapter 9

Conclusion

The goal of this thesis was to create IoT gateway with integrated detection modules for IoT wireless networks. Therefore I got to know the structure and functioning of IoT networks, studied the widely used IoT protocols, and examined their security weaknesses. Besides, I analyzed the systems developed within the SIoT and BeeeOn projects. In these systems analysis, I focused mainly on the BeeeOn IoT gateway and NEMEA modules for the detection of threats in IoT networks.

Based on the acquired knowledge, I created a proposal of connecting the mentioned systems to develop a functional sample of secured IoT gateway for wireless IoT protocols. I consulted this proposal with SIoT project developers as well as BeeeOn project developers. Subsequently, I integrated the systems and created the mentioned functional sample designed to run on the Turris Omnia router and thus fulfilled the primary goal of this work. The resulting system is modular and well configurable. Moreover, I created integration tests, which serve to verify overall system functionality. The implementation of the system, integration tools, and tests was created concerning the future extension of new functions and features.

Moreover, I introduced a new methodology on integrated and structured development of new modules for securing IoT networks within the gateway and on how to create new releases and versions of the gateway within the SIoT project. New tools have been created for automatic testing of the system and also instructions for commissioning the system from any available securing modules version. Furthermore, the existing BeeeOn Gateway IoT data collection module was modified to be suitable for incorporation into the BeeeOn official repository.

The overall functionality of the resulting system was tested both experimentally and by implemented integration tests. Moreover, system parts running on the Turris Omnia were also subjected to stress tests. However, the overall system functionality is significantly dependant on the functionality of the used systems. The testing revealed many deficiencies in these systems. All these deficiencies were reported, and most of them were fixed except the errors connected with two detectors. Therefore these two detectors are not included in the resulting functional sample. Nevertheless, all the other system parts work as expected. Also, the stress tests have proven that the router is capable of running the system without any problems.

I see areas for improvement in the possibility of integrating a virtual machine that interacts with users to run directly on the router. Also, with a small modification of the detection modules, it would be possible to create automated integration tests that are

time-independent. For the more realistic verification of the system functionality, it would be relevant to create a real IoT network testbed to test the system in a real environment.

Bibliography

[1] Aras, E.; Ramachandran, G. S.; Lawrence, P.; et al.: *Exploring the Security Vulnerabilities of LoRa*. In *2017 3rd IEEE International Conference on Cybernetics (CYBCONF)*. June 2017. pp. 1–6. doi:10.1109/CYBConf.2017.7985777.

[2] *BeeeOn Gateway - Next generation of the gateway firmware*. [Online; visited 02.03.2019].
Retrieved from: https://github.com/BeeeOn/gateway

[3] Bon, M.: *A Basic Introduction to BLE Security*. [Online; visited 01.03.2019].
Retrieved from: https://www.digikey.com/eewiki/display/Wireless/A+Basic+Introduction+to+BLE+Security

[4] Cejka, T.; Bartos, V.; Svepes, M.; et al.: *NEMEA: A framework for network traffic analysis*. In *2016 12th International Conference on Network and Service Management (CNSM)*. IEEE. 10 2016. pp. 195–201. doi:10.1109/CNSM.2016.7818417.

[5] Cisco: *Fog Computing and the Internet of Things: Extend the Cloud to Where the Things Are*. 2015. [Online; visited 20.12.2018].
Retrieved from: https://www.cisco.com/c/dam/en_us/solutions/trends/iot/docs/computing-overview.pdf

[6] Fouladi, B.; Ghanoun, S.: *Security Evaluation of the Z-Wave Wireless Protocol*. [Online; visited 29.12.2018].
Retrieved from: https://sensepost.com/cms/resources/conferences/2013/bh_zwave/Security%20Evaluation%20of%20Z-Wave_WP.pdf

[7] Jasek, S.: *GATTACKING BLUETOOTH SMART DEVICES*. [Online; visited 29.12.2018].
Retrieved from: http://gattack.io/whitepaper.pdf

[8] Krejci, R.; Hujnak, O.; Svepes, M.: *Security survey of the IoT wireless protocols*. In *2017 25th Telecommunication Forum (TELFOR)*. IEEE. 11 2017. ISBN 978-1-5386-3073-0. pp. 1–4. doi:10.1109/TELFOR.2017.8249286.

[9] Kwon, G.; Kim, J.; Noh, J.; et al.: *Bluetooth low energy security vulnerability and improvement method*. In *2016 IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia)*. IEEE. 10 2016. ISBN 978-1-5090-2743-9. pp. 1–4. doi:10.1109/ICCE-Asia.2016.7804832.

[10] LoRa Alliance Technical Committee: *LoRaWAN^{TM} 1.1 Specification*. 2017. [Online; visited 13.02.2019].

Retrieved from: https://lora-alliance.org/sites/default/files/2018-04/lorawantm_specification_-v1.1.pdf

[11] Mackensen, E.; Lai, M.; Wendt, T. M.: *Bluetooth Low Energy (BLE) based wireless sensors*. In *2012 IEEE Sensors*. IEEE. 10 2012. ISBN 978-1-4577-1767-3. pp. 1–4. doi:10.1109/ICSENS.2012.6411303.

[12] *NEMEA modules for securing IoT networks*. May 2019. [Online; visited 01.03.2019]. Retrieved from: https://github.com/CESNET/NEMEA-SIoT

[13] Rosa, T.: *Bypassing Passkey Authentication in Bluetooth Low Energy*. 2013. [Online; visited 26.01.2019].
Retrieved from: http://eprint.iacr.org/2013/309

[14] Ryan, M.: *Bluetooth: With Low Energy comes Low Security*. [Online; visited 25.01.2019].
Retrieved from:
https://www.usenix.org/system/files/conference/woot13/woot13-ryan.pdf

[15] Sankaran, S.: *Modeling the performance of IoT networks*. In *2016 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)*. IEEE. 11 2016. ISBN 978-1-5090-2193-2. pp. 1–6. doi:10.1109/ANTS.2016.7947807.

[16] Soukup, D.: *Detekce anomálií v provozu IoT sítí*. Master's Thesis. České vysoké učení technické v Praze, Fakulta informačních technologií. Praha. 2018.

[17] Statista: *Internet of Things (IoT) connected devices installed base worldwide from 2015 to 2025 (in billions)*. [Online; visited 20.12.2018].
Retrieved from: https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/

[18] Tierney, A.: *Z-Shave - Exploiting Z-Wave downgrade attacks*. [Online; visited 25.01.2019].
Retrieved from: https://www.pentestpartners.com/security-blog/z-shave-exploiting-z-wave-downgrade-attacks/

[19] Yaakop, M. B.; Malik, I. A. A.; bin Suboh, Z.; et al.: *Bluetooth 5.0 throughput comparison for internet of thing usability a survey*. In *2017 International Conference on Engineering Technology and Technopreneurship (ICE2T)*. IEEE. 9 2017. ISBN 978-1-5386-1805-9. pp. 1–6. doi:10.1109/ICE2T.2017.8215995.

[20] Yang, X.: *LoRaWAN: Vulnerability Analysis and Practical Exploitation*. Master's Thesis. Delft University of Technology. Delft. 2017.

[21] Yassein, M. B.; Mardini, W.; Khalil, A.: *Smart homes automation using Z-wave protocol*. In *2016 International Conference on Engineering & MIS (ICEMIS)*. IEEE. 9 2016. ISBN 978-1-5090-5579-1. pp. 1–6. doi:10.1109/ICEMIS.2016.7745306.

[22] Z-Wave Alliance: *2018 END OF YEAR Z-WAVE ECOSYSTEM REPORT*. [Online; visited 26.01.2019].
Retrieved from: https://z-wavealliance.org/wp-content/uploads/2019/01/Z-Wave-Alliance-End-of-Year-Report-FINAL-for-web.pdf