


12-1-2016

Vulnerability Analysis and Security Framework for Zigbee Communication in IOT

Charbel Azzi

University of Nevada, Las Vegas, azzic@unlv.nevada.edu

Follow this and additional works at: <https://digitalscholarship.unlv.edu/thesesdissertations>

 Part of the [Computer Engineering Commons](#), [Computer Sciences Commons](#), [Electrical and Computer Engineering Commons](#), and the [Library and Information Science Commons](#)

Repository Citation

Azzi, Charbel, "Vulnerability Analysis and Security Framework for Zigbee Communication in IOT" (2016). *UNLV Theses, Dissertations, Professional Papers, and Capstones*. 2849.

<https://digitalscholarship.unlv.edu/thesesdissertations/2849>

This Thesis is brought to you for free and open access by Digital Scholarship@UNLV. It has been accepted for inclusion in UNLV Theses, Dissertations, Professional Papers, and Capstones by an authorized administrator of Digital Scholarship@UNLV. For more information, please contact digitalscholarship@unlv.edu.

VULNERABILITY ANALYSIS AND SECURITY FRAMEWORK FOR
ZIGBEE COMMUNICATION IN IOT

By

Charbel Azzi

Bachelor of Science - Computer Engineering

University of Nevada, Las Vegas

2011

A thesis submitted in partial fulfillment

of the requirement for the

Master of Science – Computer Science

Department of Computer Science

Howard R. Hughes College of Engineering

The Graduate College

University of Nevada, Las Vegas

December 2016



Thesis Approval

The Graduate College
The University of Nevada, Las Vegas

November 29, 2016

This thesis prepared by

Charbel Azzi

entitled

Vulnerability Analysis and Security Framework for Zigbee Communication in IOT

is approved in partial fulfillment of the requirements for the degree of

Master of Science – Computer Science
Department of Computer Science

Yoohwan Kim, Ph.D.
Examination Committee Chair

Kathryn Hausbeck Korgan, Ph.D.
Graduate College Interim Dean

Ajoy Datta, Ph.D.
Examination Committee Member

Laxmi Gewali, Ph.D.
Examination Committee Member

Emma E. Regentova, Ph.D.
Graduate College Faculty Representative

Abstract

Securing IoT (Internet of Things) systems in general, regardless of the communication technology used, has been the concern of many researchers and private companies. As for ZigBee security concerns, much research and many experiments have been conducted to better predict the nature of potential security threats. In this research we are addressing several ZigBee vulnerabilities by performing first hand experiments and attack simulations on ZigBee protocol. This will allow us to better understand the security issues surveyed and find ways to mitigate them. Based on the attack simulations performed and the survey conducted, we have developed a ZigBee IoT framework that could be used to solve security issues in the ZigBee protocol and in IoT in general. The system developed will: (1) predict a potential malicious attack by detecting the absence of a ZigBee node in the network and responding accordingly through a notification to the user of the system, (2) add another layer of encryption to the data transmitted between the ZigBee devices, (3) provide best practices for configuring and securing ZigBee devices and network, and (4) educate consumers about privacy and data security by involving them in the installation and giving them the autonomy to track in real time any motion activities detected around their house and inputting the time period that they would be notified should any suspicious activity occur.

Acknowledgments

I would like to sincerely thank Dr. Yoohwan Kim, my research advisor, for all the support, guidance, kindness and patience that he has offered me during the cycle of developing my thesis. Dr. Kim, despite your busy schedule, thank you for being there for me every time I needed help.

I would like to thank Dr, Ajoy K. Datta, computer science graduate coordinator at UNLV, for his counseling throughout my master degree program, and for the great knowledge and growth that I gained every time I have enrolled in one of his highly competitive classes.

I would like to thank Dr. Laxmi Gewali, chair of computer science department at UNLV, for serving as one the advisory committee members for my thesis.

I would like to thank Dr. Emma Regentova, professor at the electrical engineering department UNLV, for her genuine advice regarding my master's degree path, my career path, and for serving as a graduate college representative for my thesis.

I would like to thank Dr. Pushkin Kachroo, professor at the electrical engineering department UNLV, for the great support throughout my journey at UNLV, and his trust in me and offering me an undergraduate research assistant position at Transportation Research Center (TRC) during my undergraduate studies where I have gained a lot experience and growth.

I would like to thank Dr. Alex Paz, professor at the civil engineering department at UNLV, that have offered a graduate research assistantship at TRC that have gave me a great boost to start my professional career.

I would like to thank Dr. Rama Venkat, dean of Howard R. Hughes College of Engineering at UNLV, and Mr. Stan Goldfarb for being my advisors for my undergraduate senior design project that allowed my group and I to win the grand prize at Harriet & Fred Cox Engineering Design Award in May 2011. Also, I would like to thank them for being there whenever I needed an advice in my professional journey.

Dr. Kim, Dr. Datta, Dr. Gewali, Dr. Regentova, Dr. Pushkin, Dr. Rama Venkat, and Mr. Stan Goldfarb I would like to thank you again for shaping my future, for all the courses that I have took with you in my undergraduate and graduate career at UNLV, for your dedication and passion in teaching every semester a new generation of successful engineers.

I would like to thank my manager Craig Mersereau for the experience that I gained at work in developing web applications.

I would like to thank my friends Dr. Makram Abd el Qader for being my role model and Morris Bomparolla for their continuous encouragement and support throughout my graduate studies. Lastly, I would like to thank my parents Anwar Azzi and Hiam Dagher Azzi, my brother Charles Azzi, my sister Chimene Azzi, my uncles Wissam Dagher, Bassam Dagher, Ghassan Dagher and my grandmother Laurence Dagher for their support throughout my undergraduate studies, and their encouragement to pursue my master's degree in computer science.

Table of Contents

Abstract.....	iii
Acknowledgments	iv
Table of Contents	vi
List of Tables	x
List of Figures.....	xi
Chapter1: Introduction	1
1.1 Overview	1
1.2 Motivation	2
Chapter 2: Background on ZigBee protocol and security measures	5
2.1 ZigBee Overview	5
2.2 Devices Roles in a ZigBee Network	6
2.3 ZigBee Topologies	7
2.4 ZigBee Security.....	10
2.4.1 Network Layer Security	10
2.4.2 Application(APS) Layer Security	13

2.5	Creating and Joining ZigBee Network.....	15
2.5.1	Creating and Joining an Unsecure ZigBee Network.....	15
2.5.2	Creating and Joining a Secure ZigBee Network.....	18
Chapter 3: ZigBee Vulnerabilities and Mitigation Methods		20
3.1	Overview	20
3.2	Attacks performed by compromising the keys.....	22
3.2.1	Reuse of Initialization Vector (IV) values with the same key	22
3.2.2	Acquiring ZigBee Keys	23
3.3	Attacks without compromising the keys	26
3.3.1	Replay Attacks	26
3.3.2	Denial of Service (DoS) Attacks.....	28
Chapter 4: Attacks Simulations on ZigBee		33
4.1	Overview	33
4.2	Environment Setup.....	35
4.2.1	Hardware.....	36
4.2.2	Software Implementation.....	50

4.3	ZigBee Physical Attack	54
4.3.1	Network simulation configuration	54
4.3.2	Acquiring network configuration.....	56
4.3.3	Attack Simulation	58
4.4	ZigBee Unencrypted Network Key Attack	63
4.4.1	Network Simulation Configuration.....	63
4.4.2	Attack Simulation	64
Chapter 5: Secure Implementation of IoT Framework using ZigBee protocol		67
5.1	System Overview	67
5.2	Server	74
5.2.1	REST APIs.....	75
5.3	Web Application	78
5.3.1	AngularJS and the Mode-View-Controller Structure	79
5.3.2	Implementation/Design.....	81
5.4	IoT System Secure ZigBee Configuration	87
5.5	Network Functionality and Implementation	90

5.5.1	Network Functionality Use Cases.....	93
5.5.2	Router's Implementation/ Source Code.....	97
5.5.3	Coordinator's Implementation/ Source Code	100
Chapter 6: Conclusion and Future Work.....		104
6.1	Conclusion.....	104
6.2	Future Work	105
References		106
Curriculum Vitae		113

List of Tables

Table 1: ZigBee frame counter reaching maximum value [19]	12
Table 2: ZigBee Channels and 2.4GHz Band [20]	17
Table 3: Cases of Nodes Joining a ZigBee Network [20].....	18
Table 4: XBee AT Security Commands Description [19].....	55
Table 5: Server Rest API – Save Sensor State Request Method	75
Table 6: Server Rest API – Save Sensor State Response Method	75
Table 7: Server Rest API – Get most recent light state and its date and time Request Method	76
Table 8: Server Rest API – Get most recent light state and its date and time Response	76
Table 9: Server Rest API – Email Notification Request Method	77
Table 10: Server Rest API – Email Notification Response	77

List of Figures

Figure 1: ZigBee and 802.15.4 in OSI model.....	5
Figure 2: ZigBee Star Topology [21].....	7
Figure 3: ZigBee Tree Topology	8
Figure 4: ZigBee Mesh Topology.....	9
Figure 5: ZigBee Security Model [19]	10
Figure 6: ZigBee Network Authentication and Encryption [19]	11
Figure 7: ZigBee APS Authentication and Encryption [19]	13
Figure 8: Denial of Service ZigBee Attacks at the OSI layers	29
Figure 9 : Unauthenticated Acknowledge packet	31
Figure 10: ZigBee Network Setup for Attack Simulation.....	35
Figure 11: XBee Module Series 2	36
Figure 12: XBee Xplorer Board by Sparkfun	37
Figure 13: XBee Xplorer Connected to XBee module.....	38
Figure 14: XTCU Application.....	39
Figure 15: XTCU- Discover XBee Device.....	39

Figure 16: XTCU – Configure Port Parameters	40
Figure 17: XTCU – XBee Device Discovered.	41
Figure 18: XTCU – XBee Device Parameters Configuration	42
Figure 19: XTCU - Update Firmware.....	43
Figure 20: XTCU - Communication Logs	44
Figure 21: Arduino /mega 2560	45
Figure 22: Arduino IDE	46
Figure 23: XBee Shield Module Made by SainSmart.....	47
Figure 24: XBee Module Connected to The XBee Shield.....	48
Figure 25: XBee Shield - Jumpers Settings (1).....	48
Figure 26: Jumper Settings (2)	49
Figure 27: Sharp Infrared Sensor	50
Figure 28: Sender’s Code in The Attacks Simulations.....	51
Figure 29: Receiver’s Code in The Attacks Simulations.....	53
Figure 30: Coordinator Security Parameters in ZigBee	56
Figure 31: XBee configurations of the device acquired by the attacker	58

Figure 32: A data sample of the configuration being sent from the sender (router) to the receiver (the coordinator).....	59
Figure 33: ZigBee Network Setup in Physical Attack Simulation	60
Figure 34: Packet Sample Created and Sent to The coordinator to Turn On The LED	61
Figure 35: Coordinator security configuration for simulating ZigBee unencrypted network key attack.....	63
Figure 36: Attacker’s Device Joins the ZigBee Network Setup with Unencrypted Network Key.....	65
Figure 37: ZigBee DoS Simulation Attack.....	66
Figure 38: System’s Component Diagram.....	68
Figure 39: Router Parts and Connections	69
Figure 40: Router Assembled	70
Figure 41: Coordinator Back View	71
Figure 42: Coordinator Side View.....	72
Figure 43: Web Application UI	78
Figure 44: Model View Controller	80
Figure 45: Update Sensor Status & Notification Flow - Sequence Diagram.....	84

Figure 46: Enable Notification - Sequence Diagram	85
Figure 47: Coordinator Security Configuration	87
Figure 48: Router Security Configuration	88
Figure 49: IoT System Real Time Snapshot – No Motion Detected.....	92
Figure 50: IoT System Real Time Snapshot – Motion Detected.....	93
Figure 51: Notify User in Case of Invalid Heart Beat Message – Sequence Diagram.....	94
Figure 52: Notify User in Case of Invalid Heart Beat Message – Sequence Diagram.....	95
Figure 53: Notify the User in Case of Invalid Sensor Data, and Update the Client App in Case of Valid ones – Sequence Diagram.....	97
Figure 54: Framework Router’s Source Code	99
Figure 55: Framework Coordinator’s Source Code.....	103

Chapter1: Introduction

1.1 Overview

IoT stands for Internet of Things which constitutes of physical devices such as refrigerators, cars, buildings, health monitoring systems and many others that are embedded with sensors, actuators, radio frequency identification (RFID), and software and are connected to a network such as the Internet that enables them to exchange and collect data. Internet of Things has stepped out from its infancy and is on the path of transforming our current understanding of a static Internet to a fully integrated dynamic Future Internet [1]. Experts forecasts that 6.4 billion connected devices will be in use in the world in 2016, that is 30% increase from 2015, and will reach approximately 20 billion by 2020 [2].

Bluetooth, ZigBee, Z-Wave, 6LowPan, WiFi, GSM/3G/4G, LoRa, Neul, and Sigfox are all communication technologies that are used in IoT. Currently, ZigBee is the most used technology in home automation and smart lighting. ZigBee is expected to capture 34% volume share of the home automation and 29% of the smart lighting markets by 2021 [3] with Compound Annual Growth Rate (GACR) of 26% during the period 2016-2020 [4].

Seeing the fast growth of IoT usage, and ZigBee communication specifically has sparked our attention to investigate the securities concerns that the IoT industry faces.

1.2 Motivation

Securing IoT systems in general regardless of the communication technology used, has been the concern of many researchers and private companies. Symantec for example, has reported in its “Internet Security Threat Report” of 2015 that: “52 percent% of health apps - many of which connect to wearable devices - did not have so much as a privacy policy in place, and 20% sent personal information, logins, and passwords over the wire in clear text” [5]. In May of 2014, more than 90 people from 19 different countries in connection with “creepware” have been arrested by the FBI and the police - using Internet-connected webcams to spy on people [6]. Many researchers have also found that many cars, hospitals, oil grids and energy grids that are connected to an IoT are vulnerable to cyberattacks [7].

As for ZigBee security concerns, much research and many experiments have been conducted to better understand the security threats that it is susceptible to [8] [9] [10] [11] [12]. In Black Hat conference 2015 researchers demonstrated and exploited ZigBee vulnerabilities [8], also some frameworks and devices has been developed to sniff ZigBee packets and have succeeded in deploying multiple attacks [13]. Despite the fact that ZigBee protocol could be hacked in many different ways, researchers have agreed that solving the problem of security in IoT does not only depends on securing the IoT devices and their communication technology, but on securing the IoT system as whole and developing a full solution IoT framework that involve multiple layers of security [14] [15] [16].

Based on all the above information/researches in securing IoT systems, and the domination of ZigBee protocol in the home automation market, we have found that additional

security measures could be added to better secure ZigBee communication and IoT home automation in general. Therefore, we have focused this thesis on:

- 1- Surveying the ZigBee vulnerability
- 2- Performing first hand experiments and security attack simulations on ZigBee to develop a better understanding of how to detect and defeat them.
- 3- Designing and Developing an IoT home automation framework based on ZigBee protocol that solves many security concerns.

In brief, our IoT home automation system consists of motion sensors that detects the presence of objects around the house, notifies the user of such activity by email/message, and-allows the user to monitor these activities in real time through a web application. As you notice our system is not unique in its functionality and there are many such systems on the market today; however, the focus of our IoT system is not only to provide home automation to consumers, but more importantly to provide a full IoT framework to secure such automation. Our IoT ZigBee based framework is focused on:

- 1- Setting up multiple layers of defense, where multiple layers of security could be used to defend a particular risk by using additional encryption to the data transmitted between the ZigBee devices.
- 2- Educating consumers about privacy and data security [17], by giving them the autonomy to track in real time any motion activities detected around their house, and setup the time period that they should be notified of any suspicious activities that occurs.
- 3- Not using default IoT devices configurations [18] by configuring and securing ZigBee devices communication.

4- Predicting potential malicious attack, by detecting the absence of a ZigBee node in the network and responding accordingly through a notification to the user and to the systems' management team.

- Contribution:

Our thesis is organized as follow:

- In chapter 2, we give a background on ZigBee protocol and its security measures.
- In chapter 3, we survey ZigBee vulnerabilities and mitigation methods suggested by researchers.
- In chapter 4, we perform attack simulations on ZigBee protocol to learn and explore its vulnerabilities first hand.
- In chapter 5, we review our design and implementation of a secure IoT framework based on ZigBee protocol, to solve some of the problems discovered in the ZigBee protocol, and in any IoT system in general.
- In chapter 6, we summarize the work accomplished in this thesis and provide a roadmap of future work that provides more security measures.

Chapter 2: Background on ZigBee protocol and security measures

2.1 ZigBee Overview

ZigBee is based on the IEEE 802.15.4 standard specification and is created by a set of companies which form the ZigBee Alliance to provide an open-source wireless networking standard aimed at monitoring, and controlling services applications where responsiveness is more important for real-time operations than high bandwidth. Most ZigBee radio transceivers operate in the same 2.4GHz ISM (industrial, scientific & medical) band as Wi-Fi and Bluetooth [19]. This standard defines a communication layer level 3 and higher in the OSI model with the purpose of defining extra communication features such as authentication, encryption, and association in the upper layers application services as shown in Figure 1. To be more specific ZigBee implements two extra security layers on top of the IEEE 802.15.4 standard: The Network and Application security layers [20].

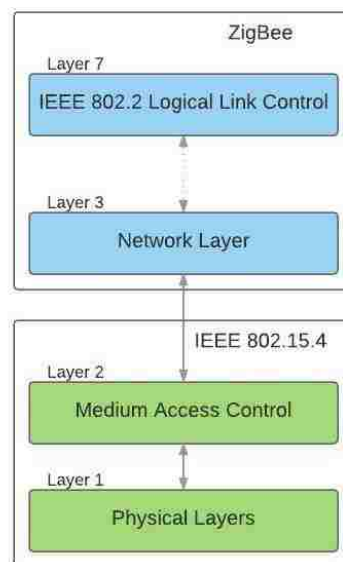


Figure 1: ZigBee and 802.15.4 in OSI model

2.2 Devices Roles in a ZigBee Network

ZigBee protocol defines three types of logical devices and each type has a specific role:

- **Coordinators:**

Are responsible for setting up the network parameters, e.g. selecting the network encryption key, and creating the network. Only one coordinator should exist in the network. The coordinators usually act as a trust center and they are responsible to set the trust center link key, and to manage and authenticate any new node that wants to join the network. Coordinators can act also as routers or end devices. The coordinators cannot be put to sleep and cannot be battery powered because the packets have to be saved to be sent to end devices and routers.

- **Routers:**

Are usually the intermediates nodes and route the information sent by end devices to the coordinator; in some case routers could also act as an end device. The routers, like the coordinators cannot be put to sleep or powered by batteries.

- **End Devices:**

Are usually the sensor nodes that collect environment data, e.g. thermostat, motion detectors, and infrared devices. End devices can either communicate with the coordinator or the routers, but not among each other. To save power, end devices can optionally be battery powered and put to sleep whenever activities are not detected for a certain period of time; similarly, end devices are awakened whenever an environment activity is detected.

2.3 ZigBee Topologies

ZigBee supports only a star, tree and a mesh network topology, but does not support cluster tree topology like 802.15.4.

- **Star topology:**

In a star topology the coordinator is placed at the center of the network, where routers and end devices can connect to. In this topology there's no differentiation between routers and end devices; routers are treated as end devices, and the coordinator is responsible of routing packages. Devices can only communicate with the coordinator and not among each other, and any packet exchange has to go through the coordinator as shown in Figure 2 [21].



Figure 2: ZigBee Star Topology [21]

- **Tree topology**

In a tree topology the coordinator acts as the tree node, where routers and/or end devices can connect and communicate to, as shown in Figure 3. A router can be a parent node and can connect to another router, coordinator, and an end device. An end device can only be a children node of the parent coordinator (root node), or to a router; also end device cannot communicate

and connect to each other. Children of any specific node can communicate among each other, or to other nodes only through their parent node. The drawback of a tree topology is that in case of a malfunctioning or disabled parent node, the children cannot communicate with other devices in the network.

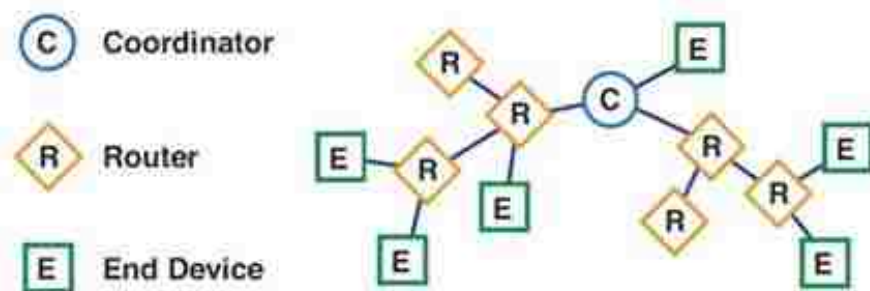


Figure 3: ZigBee Tree Topology

- **Mesh Topology**

In a mesh topology any node can send data to any other node in the network as shown in Figure 4; if the destination node is not in range, the packet will be sent to a neighboring node which will forward it to the destination node; hence it's called a self-healing topology. This topology requires one coordinator, one or multiple routers, and the end devices could be optional since the routers can act as end devices. Applying mesh topology in a large network causes a big overhead, and it is usually more complex to setup.

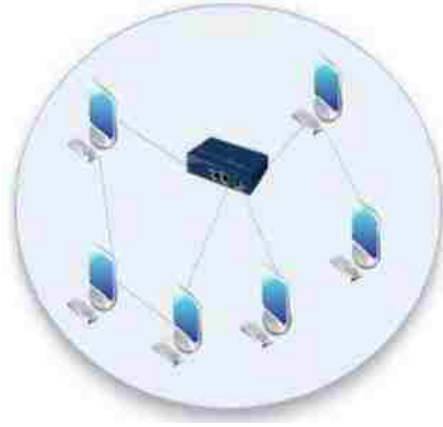


Figure 4: ZigBee Mesh Topology

2.4 ZigBee Security

ZigBee security is applied to the Network and Application layers where packages are encrypted with 128-bit AES (Advanced Encryption Standard) as shown in Figure 5 [19]. Data is encrypted by using a network encryption key and possibly a link encryption key. Devices has to have the same keys to be able communicate among each other in the network.

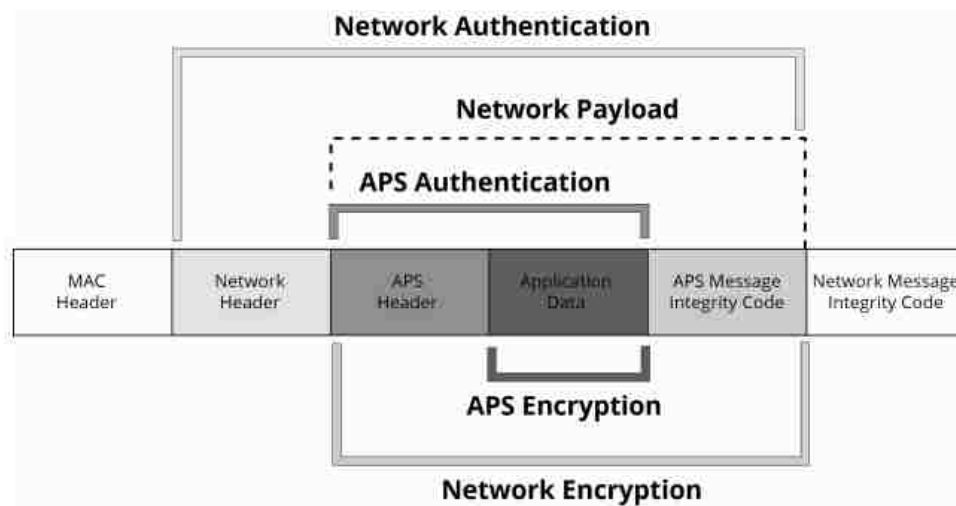


Figure 5: ZigBee Security Model [19]

2.4.1 Network Layer Security

The network layer security is implemented by using a network key to secure broadcast communication by encrypting the APS layer and application data as shown in Figure 6. Once security is enabled in the network, all data packages are encrypted with the network encryption key [19]. Security at the network layer applies to all packages transmitted and is encrypted and

decrypted on in each node of the network (hop-by-hop); however, this security does not apply to the medium access layer communication, such as beacon messages [19].

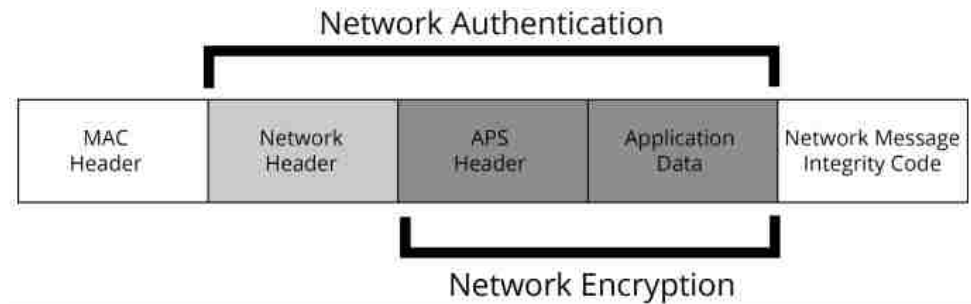


Figure 6: ZigBee Network Authentication and Encryption [19]

- **Replay Protection:**

Each node in the ZigBee network contains a 32-bit frame counter and it is located in the network header. The frame counter is incremented for every transmission to encounter replay attacks, and every node tracks the last 32-bit frame counter of each of the devices (nodes) that are connected to it. Once a device accepts a package from a neighboring node with a lesser frame counter value than it has previously received, the package will be thrown away [19]. The maximum value that a frame counter can reach is 0xFFFFFFFF, and if the maximum value is reached, no transmission can be made. Table 1 displays the time that it takes the frame counter to reach its maximum [19]. The only time the frame counter is reset to 0 is when the network key is updated.

Average Transmission Rate	Time until frame counter reach Max
1/second	136 years
10/second	13.6 years

Table 1: ZigBee frame counter reaching maximum value [19]

- **Message Authentication Code:**

Similar to IEEE 802.15.4, ZigBee uses Message Authentication Code to authenticate the network header, application header and application data. Hash code is implemented on these fields and is added to the end of the packet to form the 4-byte Message Integrity Code (MIC) that allows receiving nodes to identify that the message is not altered. When a device receives a packet and the MIC does not match its own hashed data, then it drops the packet. The MIC provides message integrity in the ZigBee protocol.

- **Encryption/Decryption:**

The packets transferred in the network encryption layer are authenticated, then encrypted and decrypted at every node in the route. As soon as a device receives the packets and is not the

destination, it encrypts and authenticates the packets using its own frame counter and source address in the network header section [19]. Packets latency is lengthier in a security enabled network than in a none enabled one because the encryption and decryption mechanism is achieved at every node.

2.4.2 Application(APS) Layer Security

Application layer security is implemented by using a shared link key to secure the unicast communication between the source and the destination devices to encrypt application data as shown in Figure 7 [19]. Application security is not required, and it provides end to end security using a link encryption key that the source and destination nodes only know about, unlike Network Layer that provides a node to node basis security.

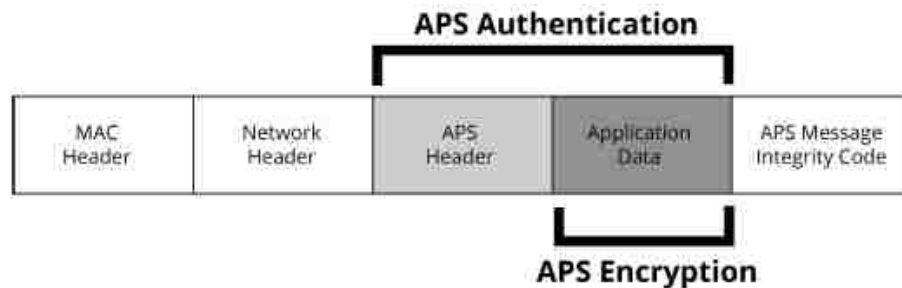


Figure 7: ZigBee APS Authentication and Encryption [19]

- **Message Authentication Code:**

Like the network layer, the application header and the application data are hashed and are appended to the packet to form a 4-byte Message Integrity Code (MIC)

- **Application link keys:**

The application security uses two different types of keys. The first is the trust center link key, where the key is recognized between the trust center and the device [19]. The trust center generates the trust center link key and is accountable for permitting nodes to join the ZigBee network. The second link key is the application link key and it is established between two devices in the network and none of these devices is a trust center.

2.5 Creating and Joining ZigBee Network

Nodes that want to join the ZigBee network, they broadcast a beacon request on a specific channel and all the routers and coordinators in the range will reply with a beacon response, that informs the new node with the Personal Area Network Identifier (PAN ID) of their operating network. If the joining node does not receive any beacon response back, then it repeats the beacon transmission on the succeeding channel until it finds a valid one. Once a valid network is found the new node transmits an association request (unicast message) informing the network his readiness to join. The router or coordinator in range transmits an association response with a 16-bit network address that has been assigned to the new joining node.

Beacon messages are not encrypted and any node can receive a beacon from a router or coordinator device; however, for a node to join the ZigBee network, its security settings has to match the security setting that are set by its parent [19] [20].

2.5.1 Creating and Joining an Unsecure ZigBee Network

ZigBee network is identified as a Personal Area Network (PAN), and is identified by distinctive PAN ID. ZigBee like 802.15.4 operates on the 2.4 GHz frequency band; it breaks the 2.4 GHz band into 16 channels. Table 2 shows the channels that ZigBee can operate on and the overlapping between the channels and the 2.4GHz band [20]. Choosing the right channel is crucial to avoid any conflict with any other protocols that uses the same channels e.g. Wi-Fi.

The coordinator is accountable of choosing the PAN ID and the channel to start a network; if they're not selected, a random PAN ID and a random channel are automatically selected. The joining device should be operating on the same preconfigured channel as the

coordinator, also the joining device should be either preconfigured with the same PAN ID as the coordinator, or it can discover the PAN ID of nearby networks for it to join; the later approach is not recommended since any device can join any network, if security is not enabled. ZigBee supports both 64-bit and 16-bit PAN ID; the 64-bit was created to resolve a potential range conflict since the 16-bit has a limited addressing space (65,535 possibilities). ZigBee stack is able to resolve a PAN ID conflict in case multiple networks choose the same PAN ID.

Decimal	Hex	Frequency	SC mask	Wi-Fi Conflict	Comments
11	0x0B	2.405GHz	0x0001	Overlaps Ch 1	Newer XBee only
12	0x0C	2.410GHz	0x0002	Overlaps Ch 1	
13	0x0D	2.415GHz	0x0004	Overlaps Ch 1	
14	0x0E	2.420GHz	0x0008	Overlaps Ch 1	
15	0x0F	2.425GHz	0x0010	Overlaps Ch 6	
16	0x10	2.430GHz	0x0020	Overlaps Ch 6	
17	0x11	2.435GHz	0x0040	Overlaps Ch 6	
18	0x12	2.440GHz	0x0080	Overlaps Ch 6	
19	0x13	2.445GHz	0x0100	Overlaps Ch 6	
20	0x14	2.450GHz	0x0200	Overlaps Ch 11	
21	0x15	2.455GHz	0x0400	Overlaps Ch 11	
22	0x16	2.460GHz	0x0800	Overlaps Ch 11	
23	0x17	2.465GHz	0x1000	Overlaps Ch 11	
24	0x18	2.470GHz	0x2000	Overlaps Ch 11	Newer XBee only
25	0x19	2.475GHz	0x4000	No Conflict	Newer XBee only
26	0x1A	2.480GHz	0x8000	No Conflict	Newer non-PRO XBee only

Table 2: ZigBee Channels and 2.4GHz Band [20]

2.5.2 Creating and Joining a Secure ZigBee Network

A secure ZigBee network requires security configuration in addition to the configuration and rules applied in an unsecure ZigBee network.

Table 3 summarizes all the cases that allow or disallow a node to join a secure or an unsecure network and to establish or disestablish the communication with other nodes.

	Joining Node Encryption Enabled	Parent Node Encryption Enabled	Network Joined	Communication Established
Case 1	NO	NO	YES	YES
Case 2	NO	YES	YES	NO
Case 3	YES	NO	NO	NO
Case 4	YES	YES	DEPENDS	DEPENDS

Table 3: Cases of Nodes Joining a ZigBee Network [20]

Table 3 shows that even if both parent node and the joining node has encryption enabled, the joining node might not be able to join the network or establish communication. This case is described as follows:

The joining node will join the network if pre-configured with the same encryption key as the parent node. The coordinator will not only be responsible for selecting the PAN ID, and the channel to operate on, but also selecting the network encryption key. In addition, the coordinator

that is usually the trust center has to pick the link encryption key (APS key). Both the network key and the link encryption key could be pre-configured or chosen randomly.

In a secure ZigBee network, any device that wants to join the network has to get the network key. If a joining device is configured with the same link key provided by the coordinator or the trust center, then the network encryption key is sent to the joining node encrypted with the link encryption key [19] [20]. In any other case the trust center will decide whether or not to send the network key to the joining node; if the trust center decides to send the network key to the joining node then the key will be sent unencrypted as a “plain text” [19] [20].

Chapter 3: ZigBee Vulnerabilities and Mitigation Methods

3.1 Overview

Securing IoT has been the main goal and the main investment of many companies and researchers, and especially that the IoT sector have experienced many real attacks that have affected large companies in different sector. Recently, about a week ago, a denial of service attack took down the internet for most of the Eastern states. The attack was aimed at Dyn, an internet and DNS provider for many big companies such as Spotify, Reddit, and New York times. The main attack was due to a malware that affected IoT devices such as webcams, DVRs, and routers [22]. Once these IoT devices are infected they become part of botnet army that drives a huge number of malicious traffic to a specific target. This type of attack is not new or a surprise to many researchers in the IoT field, but in the contrary it has been proven and predicted by many researchers that have tested a variety of IoT devices.

Considering the importance of the security in IoT devices, we discuss in this chapter the vulnerabilities in ZigBee communication protocol and the mitigation methods that has been researched and proven by many researchers. We have divided ZigBee vulnerabilities into two categories: 1) vulnerabilities by compromising the keys, and 2) vulnerabilities without compromising the keys. In each of these categories we go over scenarios and methods that could expose ZigBee to malicious attacks, and we suggest mitigation methods for each one of them. Despite the fact of the existence of such vulnerabilities in ZigBee, we were not able to find any real world attack that specifically mention that the cause was due to the ZigBee protocol. Also, most of the real world attacks found did not mention any specific, and known communication

standard that IoT uses; however, we have provided real world IoT attack scenarios that have occurred and are related to the same types of vulnerabilities that ZigBee suffers from. Even though the articles about the real world malicious attacks have undisclosed the communication protocols, we have correlated each one of them to the vulnerabilities in ZigBee to emphasize the fact that such attacks are real and their occurrence is very likely to occur in ZigBee devices and networks.

3.2 Attacks performed by compromising the keys

3.2.1 Reuse of Initialization Vector (IV) values with the same key

Reusing Initialization Vector (IV) value with the same key is a security vulnerability inherited by ZigBee from 802.15.4. Advanced Encryption Standard Counter (AES-CTR) mode. Reusing nonce values such as Initialization Vector (IV) with the same key introduces vulnerabilities because an attacker can recover two plaintexts using their ciphertexts in the AES-CTR mode [23]. Recovering the two plaintexts is a simple operation could be performed by an attacker, and that is by using XOR operation on the two ciphertexts that has been encrypted with the same keys and nonce values, which is similar to the attack that can be performed on WEP [24] [23]. For example, if for any reason the device sent two consecutive messages M_1 and M_2 encrypted with the same nonce value and the same key, then by capturing the data transmitted and using XOR operation on C_1 and C_2 we will be able to recover partial information about the original text using the formula [25]:

$$C_1 \oplus C_2 = [M_1 \oplus E(\text{key, nonce})] \oplus [M_2 \oplus E(\text{key, nonce})] = M_1 \oplus M_2$$

Below are 2 scenarios that allows the reuse of IV values with the same key:

- **Multiple Independent Access Control Entries**

802.15.4 radio devices have an Access Control List (ACL) for configuring the security suite that should be used. Since each chip has multiple independent ACL, the same IV could be reused with the same Key causing vulnerability in the network.

- **Repopulating Access Control Table**

Another scenario that could happen is when a ZigBee device goes to sleep (or loses power), the ACL entries are lost and need to be repopulated; at this time if the application layer regenerates the ACL table it may also regenerate the same IV values with the same keys that has been used before the device loses power [23] [26]. For example, in one of the researchers' simulation, they implemented this attack by taking out the battery of the device every time one message was sent, and they used XOR operation on both the encrypted message and obtained the original data transmitted [25].

➤ **Best Practices of avoiding the reuse of IV**

A solution for this flaw is to generate a new key after power disruption or a key counter should be stored in flash memory so that they are not lost and recover them after each power failure [25] [23]. However, keys establishment becomes an issue, every time a new key is generated, also maintaining the ACL entries in a sleepy mode requires power which is another issue, since ZigBee is designed for low power consumption. An alternative solution is to store the ACL entries in the upper layer, for all sleepy devices [23].

3.2.2 Acquiring ZigBee Keys

The network encryption key main functionality in ZigBee is to encode broadcast messages, and is shared among all devices; it suffices to compromise one node in order to compromise the whole network. The link key in ZigBee is used to encrypt unicast messages between two nodes; therefore, when a node is compromised, a hacker can obtain all the unicast communication of the that device. ZigBee security is based on the assumption that keys are stored securely, and the coordinator is preconfigured with the network key and other devices

such as the routers and end devices are pre-configured with the link encryption key and obtain the network encryption key over-air.

Below are examples of acquiring ZigBee Keys:

- **Physical Attacks**

ZigBee network or link key can be obtained by a physical attack. The keys can be extracted from ZigBee devices' flash memory once a physical access is achieved. Also when a device is removed from the network, ZigBee does not invalidate the keys and generate new ones and that allow tempering the whole network [27]. Several researchers that gained physical access to the ZigBee device have extracted the firmware and found the encryption from there [28] [25].

Millions of IoT devices use the same cryptographic secrets key; SEC have analyzed more than 4,000 IoT devices in the market from over 70 vendors and extracted the encryption keys from the firmware, and have found that most of the devices use the same keys. The number of unique keys was 580, and out of these 230 were actively used [29].

- **Network key sent unencrypted over the air**

In ZigBee protocol if the link key is not configured by the coordinator, then the network key will be sent unencrypted to the joining devices that match the PAN ID and the channel that the network is operating on. The PAN ID is a 64-bit value, and will take some processing time to find the right PAN ID, but eventually an attacker can find its value. Also, the channel numbers that ZigBee operates on is limited to 16 as shown in Table 2 which makes it easy to find the correct network channel [19]. For example, Vidgren, et al. [30] demonstrate an attack on a

ZigBee network that sends the network key unencrypted over air where they gain access to the network key by intercepting the ZigBee traffic.

- **Default Link Key values**

Default link key values are used by manufacturers to provide interoperability for All ZigBee devices. For example, in a case where a user has bought a new ZigBee device to add it to his/ her own home automation network and that new device is not recognized or it's unknown or has no specific authorization associated with it, the network will allow a default link key to fall back to at the startup time [8]. Therefore, an attacker can join the network by using an unknown type device and collect the data needed to temper the network's functionality. SEC Consult's research has also revealed that millions of IoT devices are directly accessible via the Internet, and that is due to insecure default configuration [29].

- **Best Practices to prevent acquiring ZigBee Keys**

In order to prevent the acquisition of ZigBee keys by an attacker, the keys must be preloaded out of band and not transmitted over the air, and ZigBee devices physical location should be secured at all-time. Olawumi et al. [10] suggest that the Standard Security level (sending the network key unencrypted over air) should be removed all together from the ZigBee protocol. Also default configurations of keys or a fall back default keys should not be allowed by the manufacturers [8].

3.3 Attacks without compromising the keys

3.3.1 Replay Attacks

Replay attacks mean that an attacker can sniff a packet, or record packets traffic in a network and sends it back at a later time to cause a malicious attack. Let's consider for example that a system consisting of two devices, where the first one is attached to a motion sensor that sends data to the second device whenever motion is detected; in this case an attacker can capture the data sent by the first device and replay it at a later time causing the second device to assume that there was a motion detected while in fact there was not any. ZigBee implements a mechanism to avoid replay attacks by using a frame counter; however, Joshua Wright, has succeeded in implementing a replay attack and has mentioned: "802.15.4 has no replay protection and ZigBee has meager replay protection" and "An attacker can replay any previously observed traffic until key rotation" [13]. Many other researchers have also succeeded in implementing the replay attack using the "Killer Bee" that was made by Joshua Wright [13] [10].

In October 2016, CERT [31] has released an article that exploited "The Animas OneTouch Ping", insulin pump vulnerabilities. The "OneTouch" uses a custom communication protocol that does not provide sufficient protection against replay attacks. "Once the attacker spoofs the meter remote, he/she could initiate commands from the remote to the pump and attempt to replay them from a device other than the meter remote to the pump" [31].

Below are 3 examples of implementing replay attack in ZigBee:

- **ZigBee without encryption**

Reply attack could easily happen in a ZigBee network where security is not enabled which leads to a ZigBee network operating without any encryption, authentication or a frame counter. In this case an attacker can sniff the packet using another ZigBee device connected to a computer and capture the packets transmitted. Since authentication and frame counter are disabled in the network, an attacker can replay the same packets, or even change the data contained in that packet and sends it using any ZigBee device causing an unaccepted behavior in the network.

- **ZigBee network without trust center**

ZigBee networks that do not use a trust center are using the same key and are unprotected from packets replays. It's sufficient for an attacker to get hold of a higher sequence number packet and at later time when the frame counter is reset it can retransmit the packet [25]. Durech et. al. [25] has implemented such attack using SmartRF Studio 7 where they captured the encrypted data that controls the lights, and at the time of the sequence number has been reset they have resent the packet and turned on the light.

- **802.15.4 Mac Layer Unauthenticated Encryption**

Replay attack could happen on the 802.15.4 Mac layer if no authentication is enabled at that layer. "An attacker can send a message with the key counter and frame counter set to maximum, even with a payload encrypted with the wrong key. In this case the mac layer will set their values to max and any further packets received will be discarded causing a denial of service (DoS)" [9].

➤ **Best Practices to avoid replay attacks**

Replay attacks on ZigBee can be avoided by enabling security and using the trust center which will allow authentication, frame counter enabling and keys rotation. Olawumi et al. [10] suggest that ZigBee alliance should consider a time stamp that should be integrated with encryption mechanism, where in case of a packet obtained by an attacker and replayed, this packet will be rejected by the receiver because the time difference. Cache et al. [32] suggested that replay attacks could be avoided by configuring the ZigBee protocol in a way that it can confirm that the sequence number of the newly received packet is at least one number greater than the sequence number of the previously received one.

3.3.2 Denial of Service (DoS) Attacks

ZigBee alliance had put a good effort to achieve authenticity and confidentiality to the communicated packets; though, denial-of-service (DoS) is still an issue and no effort has been done in this area. Multiple stack layers could be affected by this type of attack and that depends if the attacker has joined the network (insider attack) or not (outsider attack). If the attacker has joined the network, the DoS may be conducted at the physical, medium access control, network, and application layers, but in case it's an outsider the DoS could happen only at the physical and medium access control layers. Figure 8 shows the attacks at several OSI layers.

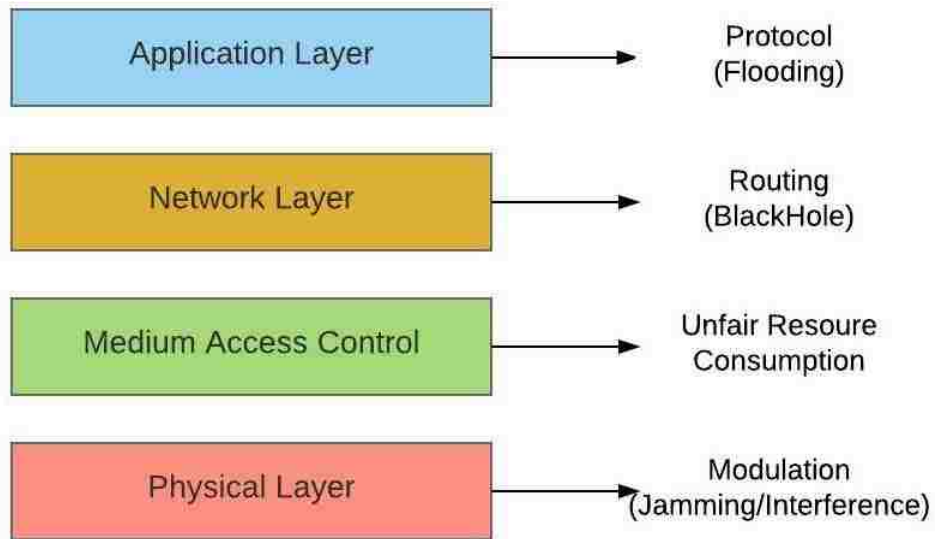


Figure 8: Denial of Service ZigBee Attacks at the OSI layers

DoS attacks are very common in the attacks related to IoT in general. Recently, in September 20th of 2016, OHV a French firm, has claimed that it was targeted by a distributed DoS attack that sent data in rate of 800 Gb/s, making it the largest known DoS attack in history, to 150,000 IP enabled internet of things devices [33].

On Oct. 21st a massive DoS attack caused an internet outage across the united states, and it was due to a botnet that was built out of IoT devices. It's confirmed that 10% of the IP-enabled cameras, DVRs, home networking gears and other connected devices compromised by Mirai were involved in that attack. The attack kept Dyn, a domain provider company and a number of its high-profile customers offline throughout different period of times during the day [22].

- **Insider Attack**

Insiders' attacks can happen at the Application Layer (APS) by flooding the network with messages. For example, an attacker can send a load of messages without any delays which causes the whole network to freeze. Also insider attacks can happen at the Network Layer (NWK), by stopping the forwarded transmission of data between devices, that can alter the routing protocol [9]. Once an attacker joins the network he basically has a complete control of almost everything in the network.

- **Outside Attack**

Outsiders' attacks can happen at the medium access layer; ZigBee uses CSMA/CA (if it is running in non-beacon mode) [9]. An attacker can send data continuously over the channel, which denies any other devices to communicate to each other. That scenario could easily happen and cause a DoS in the network since devices that use CSMA/CA communication will always back off if the channel is busy. At the physical layer, the denial of service attack could be achieved by direct jamming of the channel. Such attack can be performed by placing an outcast device close to the network that disrupts the signal by changing the Power Spectral Density (PSD) [9], [11], and [12].

In ZigBee protocol, the sender could optionally enable the ACK by setting a flag in the packet sent, the receiver then has to send a new packet containing the ACK response. Nonetheless, IEEE 802.15.4 does not have integrity checks on acknowledged packets, therefore any device can reply with an ACK packet response. For example, in a none compromised network a sender sends a packet to the receiver and waits for an ACK confirming that the

receiver has received the packet in order to continue its operation. In case an ACK was not received, the sender will send the packet again; however, in case of the network being compromised and the ACK packet is not authenticated, an attacker will intercept the packet and not allow it to be sent to the receiver and sends a fake ACK to the sender. Figure 9 describes that scenario.

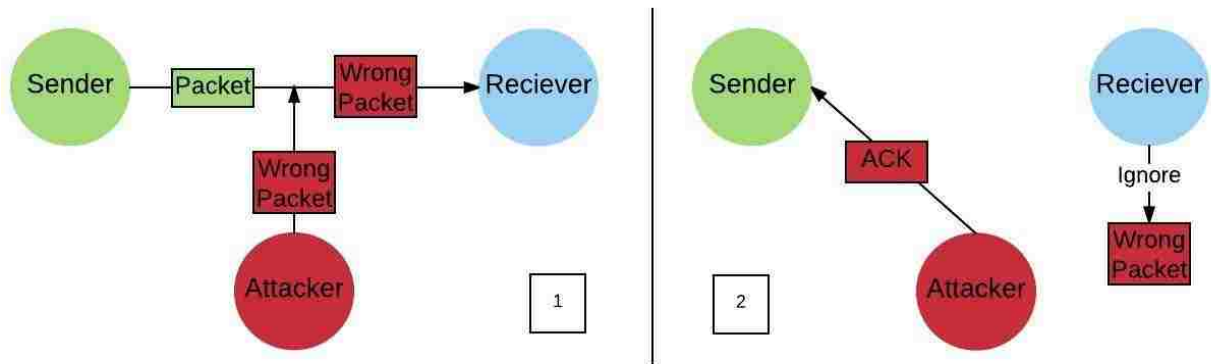


Figure 9 : Unauthenticated Acknowledge packet

Unauthenticated acknowledge packets is another “OneTouch” security vulnerability that has been exploited. An unauthenticated remote attacker can spoof acknowledgement packets to cause a remote to believe an acknowledgement was received after performing a command [31].

➤ **Best Practices to avoid DoS attacks**

Insider DoS attacks can be prevented by not allowing unauthenticated devices to join the network as described in the section “Best Practices to prevent acquiring ZigBee Keys”, and also by enabling security in the network. DoS attacks can be also avoided by placing a device that detects external signals interference close to the ZigBee network. Cao et al. [34] suggested

tracking the energy depletion of the ZigBee devices, since a DoS attack will deplete the power of the devices much faster than normal. Another, mitigation is to maintain a list of the misbehaving nodes, and if the victim node observes messages with bogus security headers, it will add the sender node to the blacklist and inform the network.

Chapter 4: Attacks Simulations on ZigBee

4.1 Overview

In this chapter we have conducted experiments and security attack simulations on ZigBee protocols to learn its vulnerabilities first hand, to demonstrate the security risks in ZigBee, and to use our learnings to create a more secure IoT ZigBee platform.

The ZigBee attacks simulated in this chapter, are based on the ZigBee security vulnerabilities that were explored in chapter 3. These simulations are based on a combination of attacks, where for example, a physical attack can lead to other various attacks such as replay and DoS attacks. We have simulated the physical attack by retrieving a node from the ZigBee network, acquiring its configuration, reading its memory, configuring an intruder ZigBee device with the same configuration obtained, and perform a replay attack on the network after listening the network traffic. We have also performed a DoS attack on ZigBee network that have granted our intruder node access to join the network by acquiring the unencrypted network key. In addition to the experience and knowledge gained about ZigBee after performing these attacks, our simulation had also demonstrated another ZigBee security architecture vulnerability that is upon removing a node from the network, the coordinator does not detect such change, and does not generate and send a new network key to the other nodes [8].

In Chapter 5, and part of our IoT ZigBee platform we suggest and implement a solution for detecting the absence of a node in the ZigBee network at the application layer without

affecting the ZigBee protocol specifications, and we configure the ZigBee network using best practices and lesson learned from our attack simulations and other research papers.

4.2 Environment Setup

In all the attack scenarios we will be using the same hardware, and software to setup the network and establish the communication, the only difference between a scenario and another is the configuration of the ZigBee devices.

To demonstrate some of the possible attacks on ZigBee protocol we have setup a system that uses ZigBee communication. The system consists of a motion detector, a light that turn on or off upon motion detection, and a dummy device acting as the compromised node in the network. The ZigBee network consists of 3 nodes: a coordinator (C) and 2 routers (R). The coordinator is labeled with a (C) letter and connected to the light, the grey color router connected to the motion detector, and the green router acting as the dummy node. The bidirectional arrows, symbolizes the bidirectional communication among the nodes as shown in Figure 10.

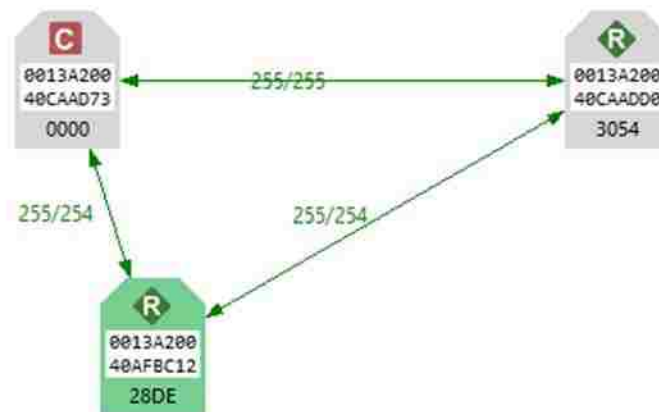


Figure 10: ZigBee Network Setup for Attack Simulation

4.2.1 Hardware

The hardware used in the simulation are chosen based on the ease of the setup and configuration. The coordinator node is formed by an Arduino Board, XBee Shield, and an XBee Pro S2 Module, the router node is formed by the same components as the coordinator with an addition to an Infrared sensor, and the dummy node is formed by an XBee module connected to an XBee Shield that is connected to the PC.

4.2.1.1 XBee Pro S2

In our attack simulations three XBee Pro S2 are used to form the ZigBee network (coordinator, and 2 routers). XBee/XBee-PRO ZB RF module is made by Digi, and it consists of ZigBee firmware loaded into XBee S2 (series 2) [19]. Figure 11 shows the XBee Series 2 module.



Figure 11: XBee Module Series 2

XBee S2 module is intended to function within ZigBee protocol. XBee S2 device is fully ZigBee compliant and can be used with any third party devices that use ZigBee protocol. XBee S1

(series 1), is a Digi product as well but should not be confused with XBee S2, since the later operates with the ZigBee firmware while the former operates with IEEE 802.15.4 protocol. XBee S1 and XBee S2 cannot communicate together since each uses a different protocol. The XBee/XBee-PRO ZB security model, network topologies, forming and joining a network is the same as ZigBee the protocol.

4.2.1.2 XBee Configuration

To configure the XBee we used the XBee Xplorer USB and the XTCU application provided by Digi.

- **XBee Xplorer USB**

XBee Explorer USB is a serial base unit for the XBee; shown in Figure 12. The XBee Xplorer is compatible with all XBee modules (S1, S2, and Pro versions). Once the XBee Xplorer is connected to the XBee, it will have a straight access to the serial and programming pins of the XBee module [35].

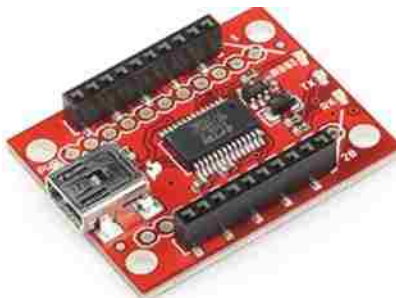


Figure 12: XBee Xplorer Board by Sparkfun

Figure 13 shows the XBee module connected to the XBee Xplorer. The XBee Xplorer connects to the PC using the USB port to allow configuration of the XBee module using the XTCU

application. In our attack simulation the XBee module connected to the XBee Explorer acts as the dummy router (grey router) node and used as the compromised node.



Figure 13: XBee Explorer Connected to XBee module

- **XTCU**

The XTCU free application provided by Digi allows an easy configuration for the XBee module, provides a graphical network view, and it is compatible with Windows and MacOS [36]. XBee module can be also configured using the AT command through the command line.

After connecting the XBee module to the XBee Explorer we connect the later to the PC using the USB port and launch the XTCU application.

- **Configuration Steps:**

Step 1: Click on the search icon at the top left of Figure 14 to search for the XBee Explorer.



Figure 14: XTCU Application

Step 2: Select the serial port that the XBee Explorer is connected to and click “Next” (in this case it’s “COM 5”); shown in Figure 15.

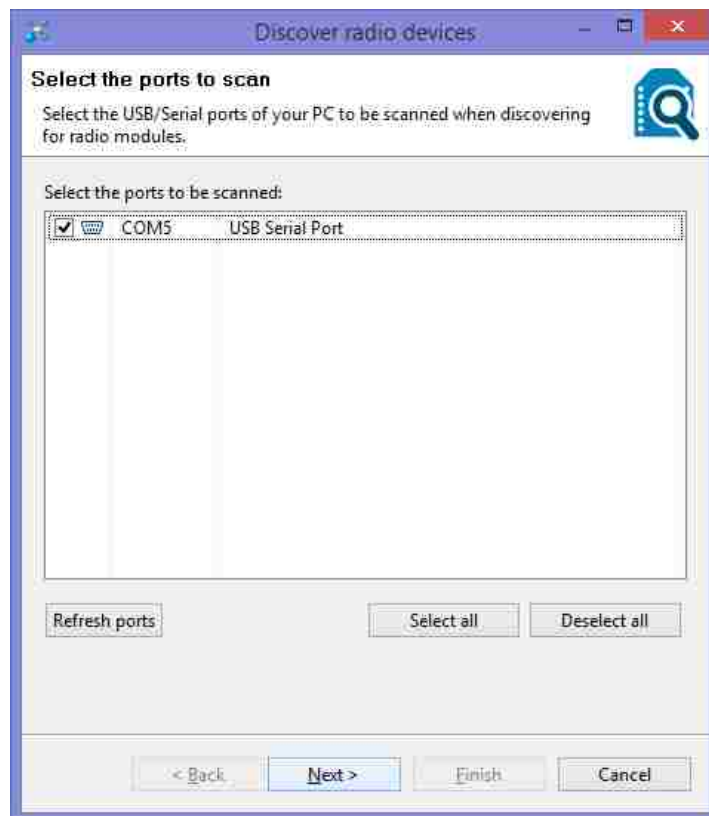


Figure 15: XTCU- Discover XBee Device

Step 3: Configure the serial port parameters and click “Finish”. In our case we left the serial port parameters in their default values as shown in Figure 16:

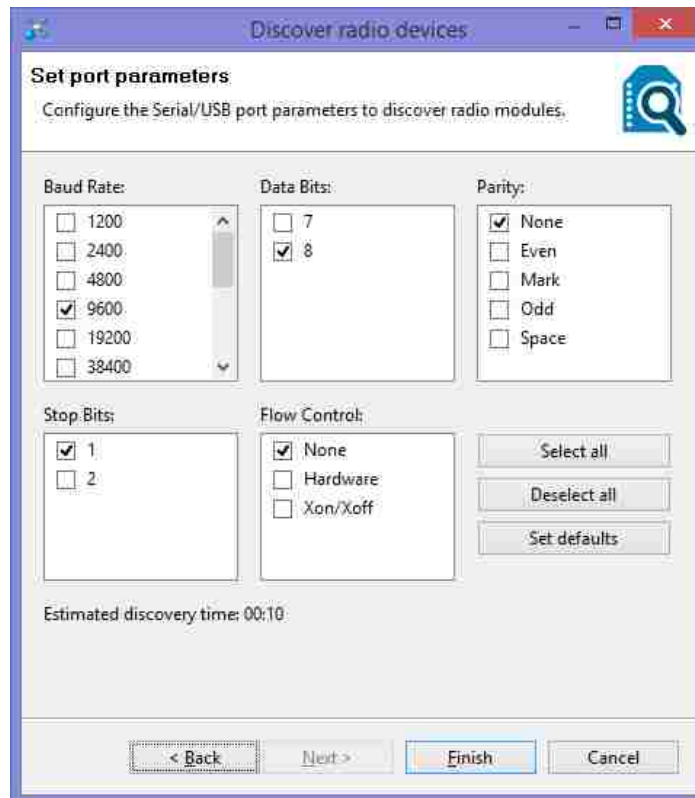


Figure 16: XTCU – Configure Port Parameters

Step 4: The XBee module is discovered by the XTCU application and some basic information of the devices are shown; such the port number that is connected to it, and the MAC Address, as shown in Figure 17. To select the device, Click “Add select devices”.

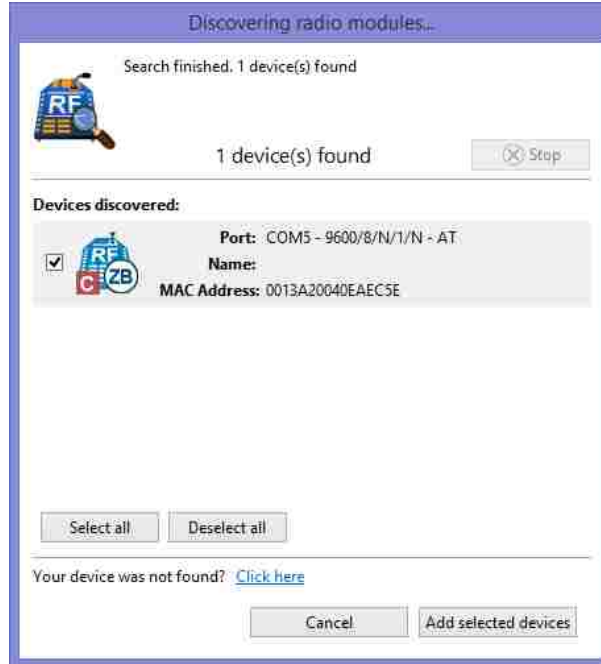


Figure 17: XTCU – XBee Device Discovered.

Step 5: Click on the device located on the left, in Figure 18, which will display all the current XBee module configuration (at the right, in Figure18). The XBee device connected has already been configured, however the parameters could be reconfigured by filling the box next to each field and clicking the “pencil” icon next to it to save/write the configuration into the XBee module.

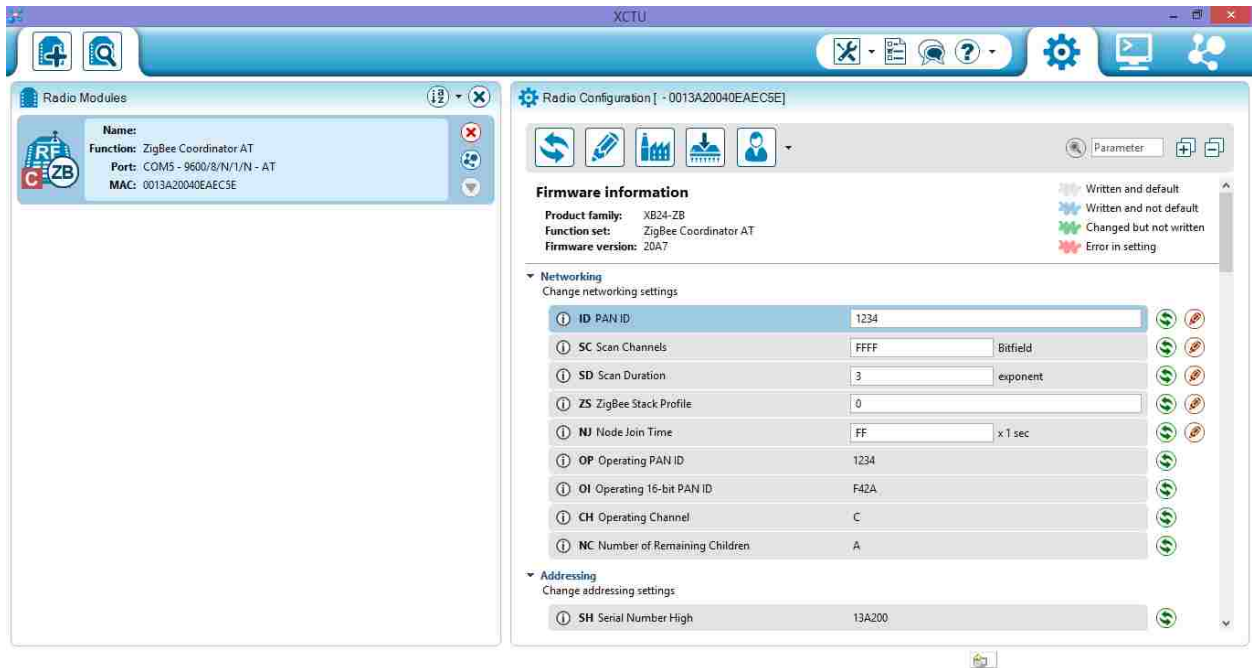


Figure 18: XTCU – XBee Device Parameters Configuration

Step 6: Besides configuring the parameter of the XBee device, the XTCU application allow to configure the function set of that device and update its firmware. For example, the same XBee device could be configured to be a coordinator and reconfigured to be a router; to do so, click on the “Chip” icon shown in Figure 18 under the radio configuration, and the “Update Firmware” window will open allowing the selection of the function set and the firmware version. After selecting the “Product Family”, the “Function Set”, and the “Firmware” version then “Update” to save the configurations, as shown in Figure 19.

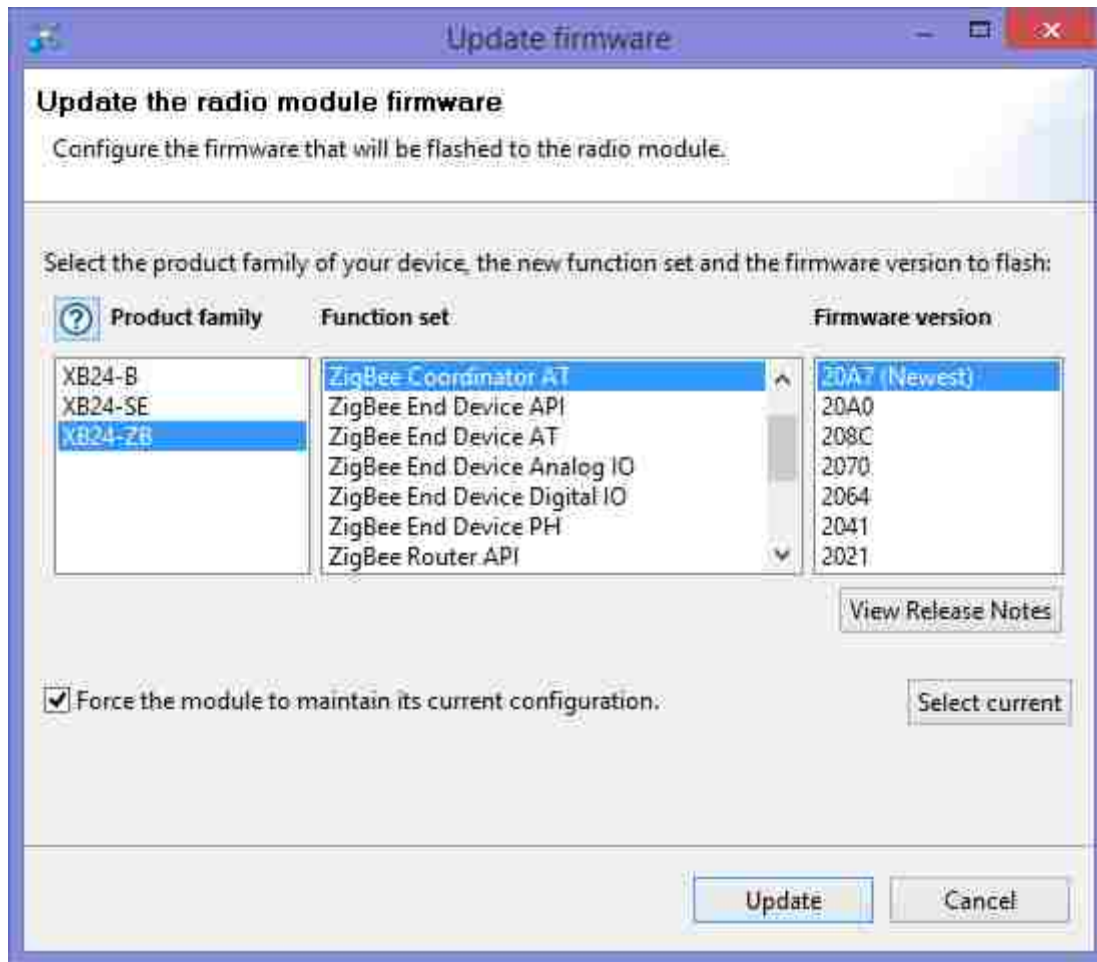


Figure 19: XTCU - Update Firmware

Step 7: Besides the configuration of the XBee module the XTCU application provide logging for the messages that being sent and received from and to the XBee module. Click on the “Monitor” at the top right of Figure 20 to see the communication log.

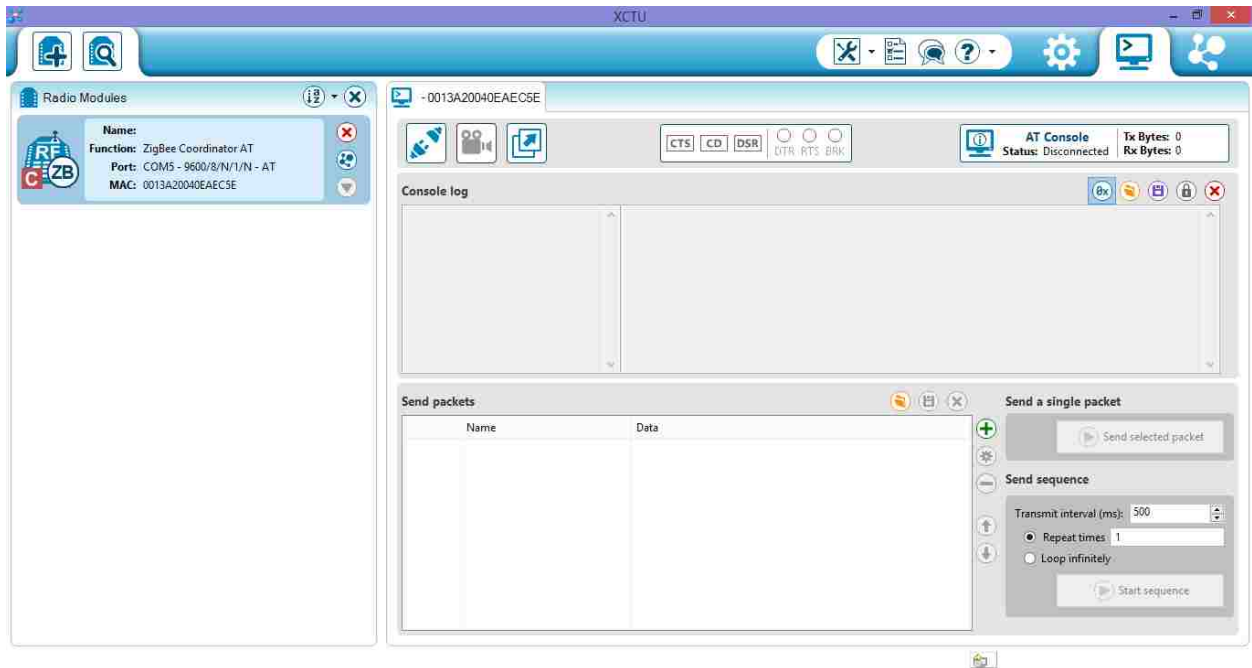


Figure 20: XCTU - Communication Logs

4.2.1.3 Arduino Board

In the attack simulation we used two Arduino boards acting as the brain of the system that analyses the incoming data and perform action upon its content (one for the coordinator, and one for the grey router shown in Figure 8). Arduino makes many types of boards; the one that we will be using in our simulation is the Arduino Mega 2560, based on the Atmega2560 microcontroller. Arduino Mega has 54 digital input/output pins, 16 analog inputs, 4 UARTs (serial ports), and 16 MHz crystal oscillator [37], shown in Figure 21.



Figure 21: Arduino /mega 2560

- **Arduino IDE:**

Arduino provides its own IDE to write, compile and upload code into their board as shown in Figure 22.

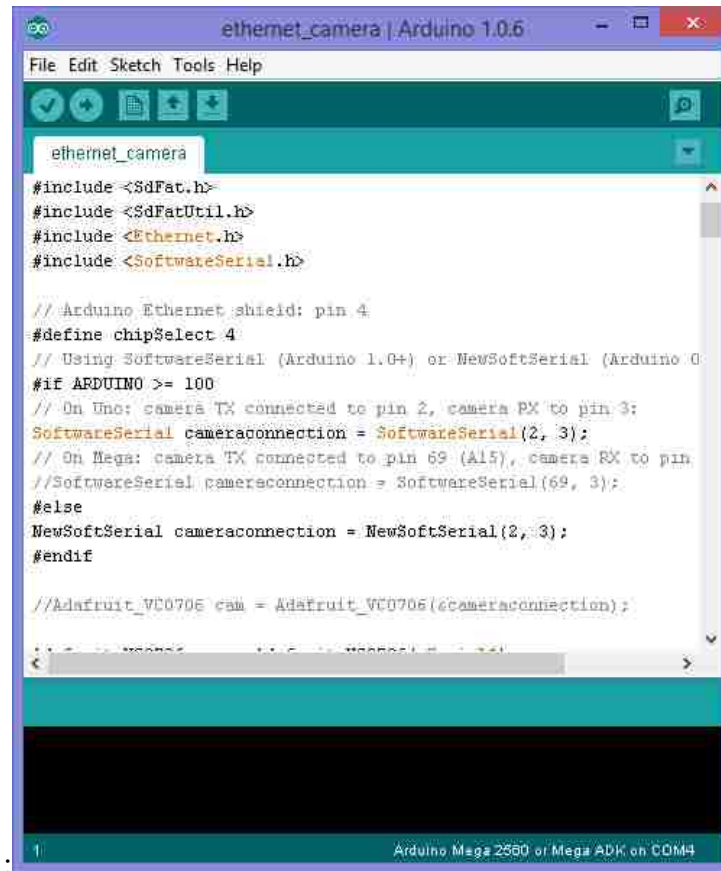


Figure 22: Arduino IDE

○ **Programing Arduino Board:**

Step 1: Connect the Arduino board to the PC using the USB port

Step 1: Open the Arduino IDE software

Step 2: Select the Arduino board. Go to Tools -> Board, and select the board in use. In our simulation we're using "Arduino Mega 2560"

Step 3: Select the serial port that the Arduino board is connected to. Go to Tools -> Serial Port

Step 4: Compile the code, by clicking the “Check” mark icon.

Step5: Program the Arduino board by clicking the “Forward” icon.

4.2.1.4 XBee Shield Module

Two XBee shield modules used for connecting the XBee Pro S2 to the Arduino boards used for the coordinator and the router. The XBee Shield module allows an easy connection between the Arduino board and the XBee module [38]. The XBee module is plugged on top of the shield and the shield is plugged into the Arduino Board allowing the data to flow between the two devices as shown in Figure 23 and 24.

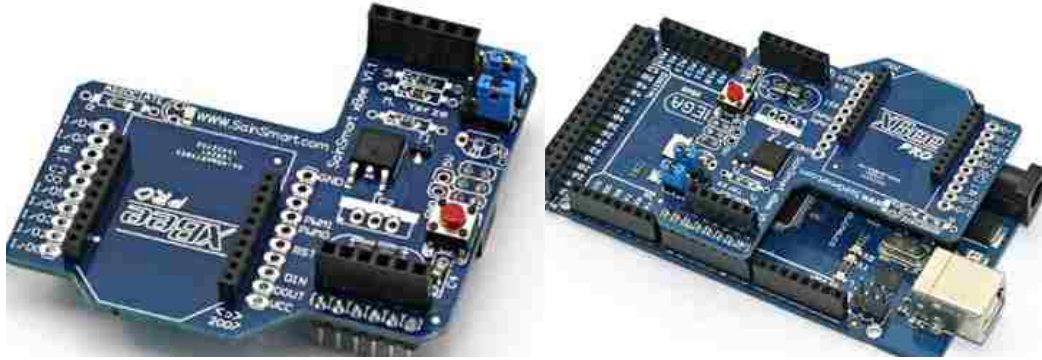


Figure 23: XBee Shield Module Made by SainSmart



Figure 24: XBee Module Connected to The XBee Shield

The XBee shield has two jumpers, that specifies: “how the XBee's serial communication links to the serial communication between the microcontroller chip and the FTDI USB-to-serial chip on the Arduino board” [37]. Upon programing the Arduino board and the XBee shield is connected to it, these 2 blue jumpers have to be moved to the right where we see the “USB” text in Figure 25.



Figure 25: XBee Shield - Jumpers Settings (1)

Once the Arduino board is programmed these 2 jumpers has to be placed to the left where the “XBEE” text is shown in figure 26, to allow the over air ZigBee communication between the XBee module and other nodes in the network.



Figure 26: Jumper Settings (2)

4.2.1.5 Infrared Sensor

One infrared proximity sensor acting as motion sensor detecting the presence of objects and it's connected to the grey router (formed by the Arduino board, XBee Shield and XBee module). The Infrared proximity sensor is made by Sharp; shown in Figure 27. “Part # GP2Y0A21YK has an analog output that varies from 3.1V at 10cm to 0.4V at 80cm” [39]. The Infrared sensor is connected to: The Arduino analog pin 8, where the voltage data is read from and converted to distance, the ground (GND) pin. and the 5V pin powered by the 5V pin on the board.



Figure 27: Sharp Infrared Sensor

4.2.2 Software Implementation

The software piece is written using the Arduino open source library and loaded into the Atmega2560 microcontroller residing in the Arduino board using the Arduino IDE. The code also includes encryption and decryption of the data that is sent over air. The encryption library, AESlib.h, uses AES 128 bit, it is implemented by Davy Landman [40].

- **Sender's code**

The sender is the Arduino Board that is connected to the grey router and to the infrared sensor. The Arduino will read the collected data by the sensor through its analog pin number 8, then it will encrypt the data using AESlib.h library, and send it serially to the XBee module (the grey router in this case) which will forward it over air to the coordinator that is connected to the Arduino receiver and to the dummy router. The sender's code is shown in Figure 28:

Sender's Code in The Attack Simulations

```
// Begin

#include <AESLib.h>

int IRpin = 8;

void setup () { Serial.begin(9600); }

void loop () {

    float volts = analogRead (IRpin)*0.0048828125;

    float distance = 65*pow (volts, -1.10);

    //encryption Key

    uint8_t key [] =

    {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};

    delay (100);

    if (distance<=150 && distance>10) {

        char data [] = "HHHHHHHHHHHHHHHHHH";

        //128 bit AES data encryption

        aes128_enc_single (key, data);

        //sending the data serially to the Xbee //module connected

        to the Arduino

        Serial.println (data);

        Delay (100);

    } else {

        char data [] = "LLLLLLLLLLLLLLLL";

        aes128_enc_single (key, data);

        Serial.println (data);

        Delay (100);

    } //End
```

Figure 28: Sender's Code in The Attacks Simulations

- **Receiver's Code**

The Receiver is the Arduino board that is connected to the coordinator and to the LED that is acting as the light. The XBee module is configured as the coordinator and will receive the data over air from the router (sender) and route the data serially to the Arduino board that is connected to it. The Arduino board in its turn will decrypt the data using the same AES library (AESlib.h), analyze it and turn on or off the LED light accordingly. The receiver's code is shown in Figure 29:

Receiver's Code in The Attack Simulations

```

const int ledPin = 53; // LED Pin

void setup() {
  Serial.begin (9600);
  pinMode (ledPin, OUTPUT);
}

void loop () {
  String content = ""; char character;
  char charBuf[50];
  uint8_t key[] = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};

  while(Serial.available()) {
    // Read data from the Xbee Module
    //(coordinator)
    character = Serial.read ();
    content.concat (character);
    delay (10);
  }

  if (content != "") {
    Serial.print ("Encrypted= ");
    Serial.println (content); //log the Encrypted //data
    recieved
    Delay (10);

    content.trim ();

    content.toCharArray (charBuf, 50) ;

    aes128_dec_single (key, charBuf); //decrypt //the data received
    Encrypted

    Serial.print ("Decrypted to bytes= ");

    Serial.println (charBuf);

    Content = charBuf;

    Serial.print ("Decrypted to strings= ");

    Serial.println (content);

    if (content ==
    "HHHHHHHHHHHHHHHHHH") {

      Serial.println ("On"); // log "On"

      digitalWrite (ledPin, HIGH); //turn LED
      on

    }

    if (content == "LLLLLLLLLLLLLLLL") {

      Serial.println ("Off"); // log "Off"

      digitalWrite (ledPin, LOW); //turn LED
      of

    }

    Delay (10);

  }

  // Code End

```

Figure 29: Receiver's Code in The Attacks Simulations

4.3 ZigBee Physical Attack

In this section we will simulate a physical attack that could happen by obtaining a ZigBee device such as the dummy node introduced in our simulation system and acquiring the configuration needed from that device by reading its memory.

4.3.1 Network simulation configuration

In this section we'll go over the coordinator security configuration only, and in the following section we'll go over the rest of the configuration. The reason for explaining only the security configuration here and no other configurations is because some of these configuration parameters are "write only" values and once they're set cannot be read using the XTCU software. Table 4 describes the AT Commands used to configure XBee devices [19].

AT Command	Name and Description	Node Type	Parameter Range	Default
EE	Encryption Enable. Set/Red the encryption enable setting.	Coordinator, Router, End Device	0 Encryption Disabled 1- Encryption Enabled	0
EO	Encryption Options. 0x0 – Send the security key unsecured over-the-air during joins 0x01 – User trust center (coordinator only)	Coordinator, Router, End Device	0-0xFF	-
NK	Network Encryption Key. Write only command; if set to 0, the module select a random network key.	Coordinator	128-bit value	0
KY	Link Key. Write Only Command. Setting KY to 0 will cause the coordinator to transmit the network key in the clear to joining devices, and will cause joining devices to acquire the network key in the clear when joining.	Coordinator, Router, End Device	128-bit value	0

Table 4: XBee AT Security Commands Description [19]

As we see in Figure 30 we set the coordinator PAN ID to "1234", the mask for the channels to scan (SC scan channels) to "FFFF", the "EE" parameter is set to "1" meaning that the security is enabled, "EO" is set to "1" so that the coordinator is used as a trust center. We have set the "KY Encryption Key" to "1234", but it could be any hexadecimal value from 0-32 characters long.

For any device to join the network, the “KY” has to be set to the same value as the coordinator, and has to be pre-configured since it cannot be sent by air. However, the “NK Network Encryption Key” is sent over air encrypted by the link key “KY” and has to only be configured by the coordinator. The network encryption key “NK” is configured to at the coordinator and set to 0, meaning that the coordinator will choose a random key that is 0-32 hexadecimal characters long [19].

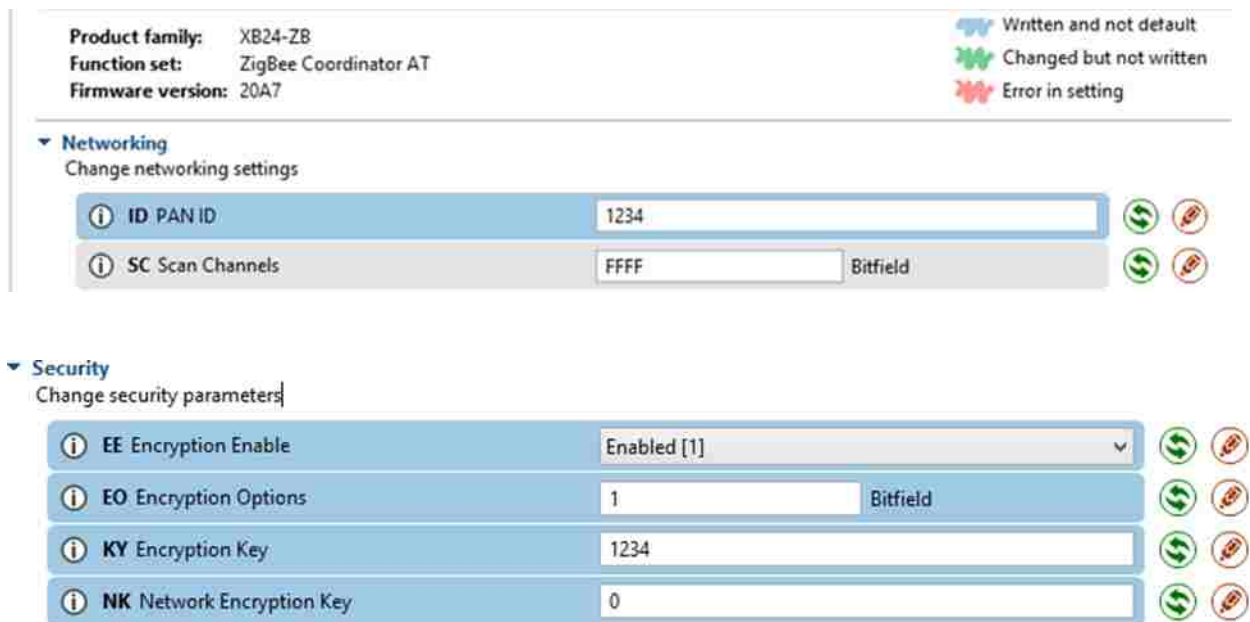


Figure 30: Coordinator Security Parameters in ZigBee

4.3.2 Acquiring network configuration

In this section we will assume that we got hold physically of the XBee dummy device and connected it to our PC using the XBee Xplorer. We used the XTCU application to read the memory of the dummy device. Once the dummy device is detected by the XTCU application, all

the parameters that resides in memory are being displayed; Figure 31 shows a small number of parameters that we chose to be displayed and are main points of interest for an attacker. These parameters are the same parameters that we have configured in the coordinator and our network is using. The most important parameters in the networking section that are read from memory are the “ID PAN ID”, the “SC Scan Channels”, and the “CH”. In the addressing section we see that an attacker can also read the Mac address of the device that is displayed in the “SH Serial Number High” and the “SL Serial Number low” fields. Also the “DH Destination Address High” and the “DL Destination Address Low” parameters are set to “0” meaning that this device is broadcasting data in the network; if these parameters were set to a value other than “0” then the node is communicating in a unicast mode and routing data to a single device in the network. The security section of the configuration is the most crucial information that we were able to read from the device memory; we can see clearly that most the security configuration that we have set in the coordinator can be read by the attacker. We were able to read the “EE Encryption Enable” that we had set to “1”; if this parameter was set to “0”, then the configuration in the networking and addressing section, are sufficient for an attacker to compromise the network. The “EO Encryption Options” is being read and set to “1”. The “KY Encryption key”, however, cannot be read by the XTCU application since it’s a “write only” value; nonetheless, other techniques have enabled researcher to get hold of the link key, and one of these techniques is the “KillerBee” application developed by Joshua Wright [13]. The link key is usually pre-configured by the manufacture and without the link key an attacker cannot configure a new XBee device and make it join the network; however, that does not stop the attacker from using the same XBee device to alter the network functionality. We also notice that the “NK network encryption key” does not exist since it’s only set at the coordinator and is sent over air.

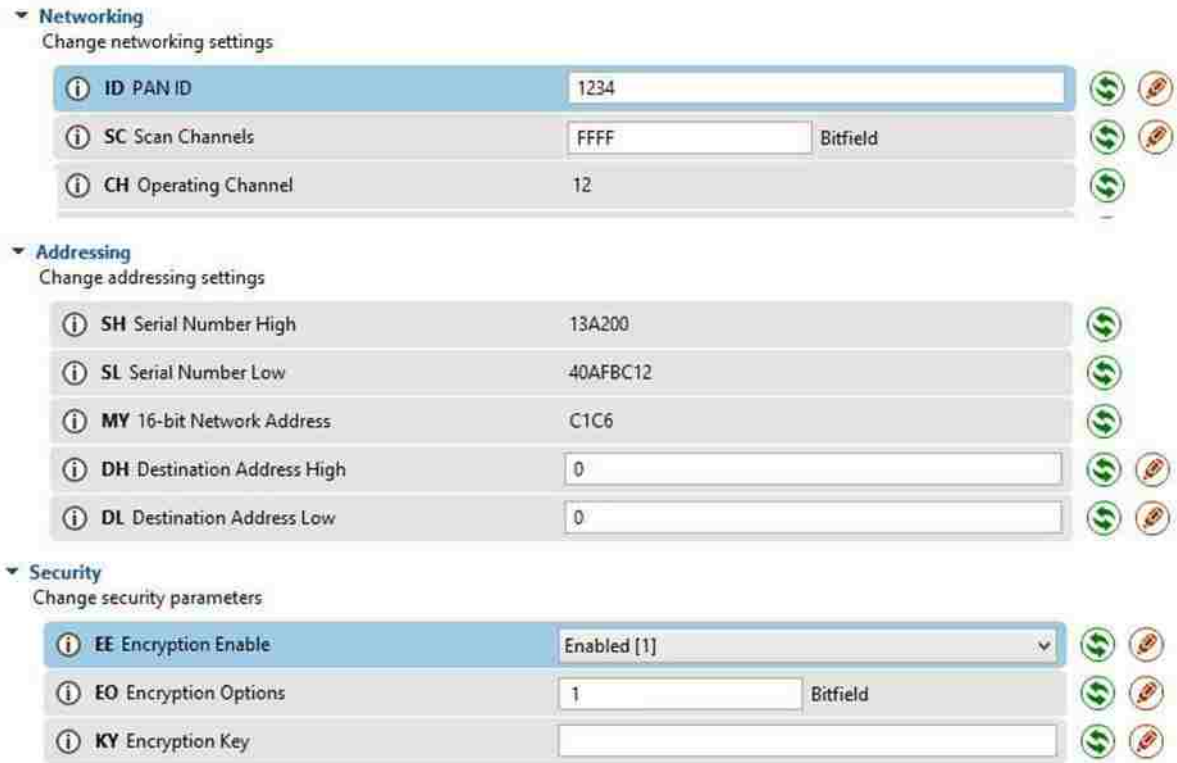


Figure 31: XBee configurations of the device acquired by the attacker

4.3.3 Attack Simulation

In the previous section we have obtained most of the configuration needed to simulate our attack on the network except the link key and the network key. Without the link we will not be able to configure the new device to join the network; however, that will not stop us or stop an attacker from compromising the network by using the same device acquired to listen and capture all the data that is being transmitted in the network and perform a replay attack.

4.3.3.1 Listening to network traffic

After acquiring the XBee dummy device and connecting it to our PC we will use the XTCU application to listen to the traffic in the network. Figure 32 shows a sample data of the messages being transmitted in the network; the following data is being transmitted from the XBee device that is connected to the sensor (the sender) and being sent to the coordinator (the receiver). As we can see in the console log, the left column contains the encrypted data “âV }C9;îÚv~wØ_.” and the right column the bytes representation of it “0A E2 56 13 7D 43 39 A1 ED DA 76 AC 77 D8 5F B7 05 0D”.

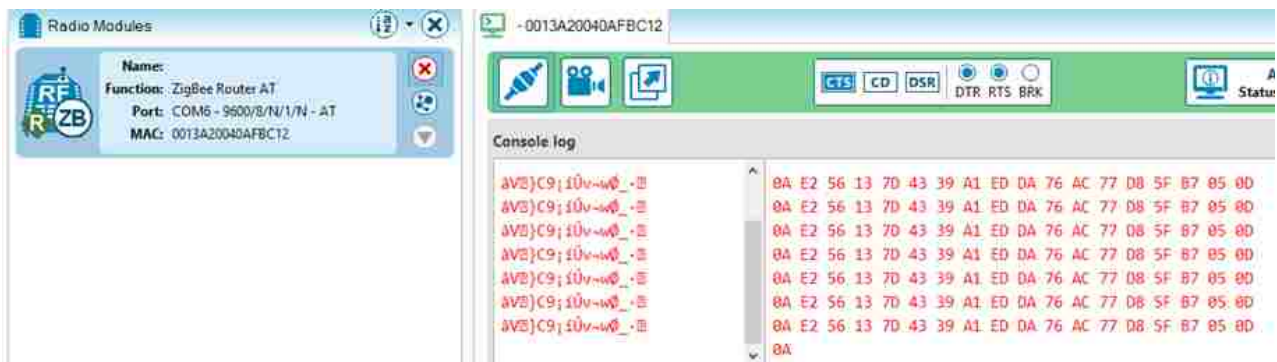


Figure 32: A data sample of the configuration being sent from the sender (router) to the receiver (the coordinator)

We were able to see data because the sender is broadcasting the data to all the nodes in the network; however, if the sender is sending the data to a specific node then we will have to perform a search on the network to detect the topology of the network and the nodes that are communicating among each other. For example, Figure 33 shows the specific details of the topology where the coordinator (C) and the green router are communicating in a bidirectional

manner while the pink router is only receiving data and not sending. The XTCU application will allow the discovery of the network topology only if the ZigBee network is configured to work in API mode and not in AT mode. In our case the XBee modules in the simulation system were configured to communicate in AT mode. In case the XBee modules were configured in API mode we will be able to detect the data being transmitted and alter the network since XTCU not only display the topology of the network but also the MAC addresses of the devices that are also the addresses of the nodes.

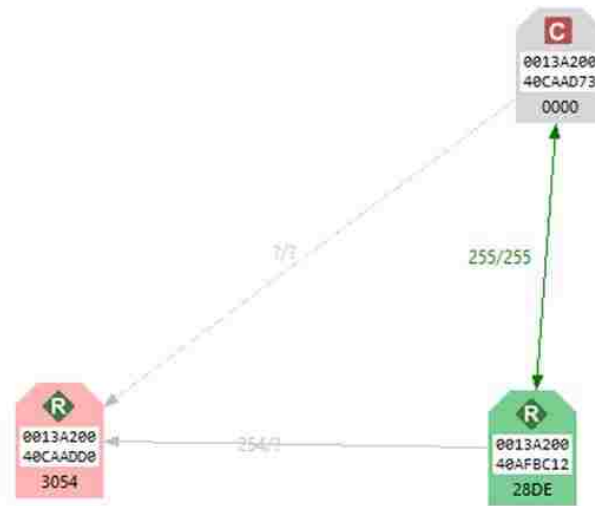


Figure 33: ZigBee Network Setup in Physical Attack Simulation

4.3.3.2 Perform replay attack

After capturing the data being transmitted we created a packet containing the same sequence of bytes that are captured and replayed it in the network; even though the data is encrypted and we were not able to decrypt it, replaying the packet in bytes was enough to trick

the coordinator and make it respond to the replayed messages. In this case two things could happen: the light on the receiver end will be turned off if it was on and vice versa. The new packet is formed by the sequence of data shown in Figure 34 “0A 76 77 BD 4C 77 F5 13 BD 5B 1A 52 2D 23 0E ED F8 0D”.

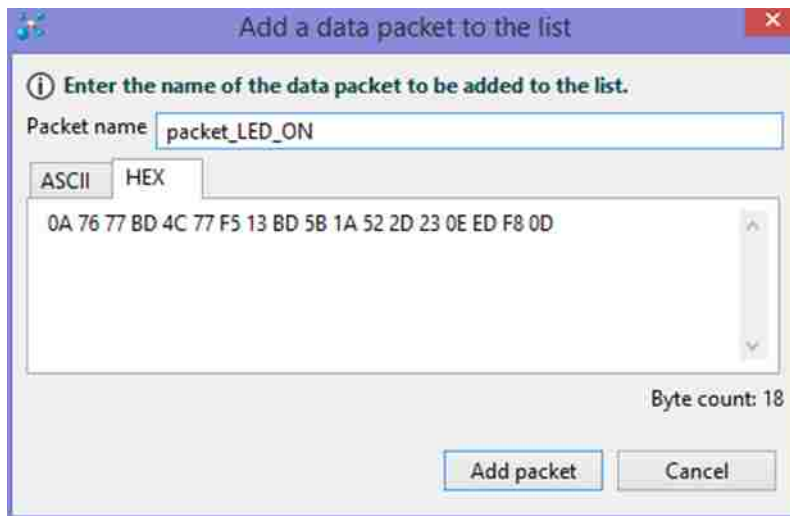


Figure 34: Packet Sample Created and Sent to The coordinator to Turn On The LED

Once the packet is formed we sent it to the network and observed that the coordinator has actually responded to it by turning the LED on while the LED should have been off since the sender in our system has not detected any motion that cause it to send the data to turn the LED on. After creating a packet to turn the LED on, we have created another packet that will turn the LED off. The packet that is being replayed to turn the LED off has the following sequence of data “0A E2 56 13 7D 43 39 A1 ED DA 76 AC 77 D8 5F B7 05 0D”. Once again we sent the fake packet over the network, and have observed the LED was turned off. Even though the LED had to stay on since an object was present on the sender side, we were able to hack its

functionality and turn it off. Luckily our system is only used for simulation and not in real world application; however, such attacks are very critical in real world applications and could lead to undesirable consequences.

4.3.3.3 Conclusion

In this section we were able to compromise the network by first getting a physical hold of the ZigBee device, acquiring the network configuration by acquiring it's the dummy node configuration, and use the device to perform a replay attack on the network and disrupt its functionalities. Our simulation had also highlighted another vulnerability in the ZigBee security architecture that upon removing a node from the ZigBee network, the coordinator does not detect such change, and does not generate and send a new network key to the other devices that are still in the network. Detecting the absence of a node in the network is crucial to prevent any stolen ZigBee device to be reused to re-join and compromise the network.

4.4 ZigBee Unencrypted Network Key Attack

In the previous section we have enabled security in the ZigBee network, and have set a link key to encrypt the network key with it. In this section, we have also enabled security in the network but this time we will not configure the link key, which forces the coordinator to send the network key (NK) unencrypted over air; using this configuration we have simulated an attack that could happen by sending the ZigBee NK unencrypted over the air to the nodes that need to join the network. In our simulation, we also used our dummy device as an external ZigBee device to join and compromise the network.

4.4.1 Network Simulation Configuration

In this simulation we have used the same configuration for the network/coordinator as in “ZigBee Physical Attack” section with the exception of not setting the link key “KY Encryption Key” to any value as shown in Figure 35. The network key “NK Network Encryption Key” will be sent encrypted over air to the devices that are interested in joining the network.

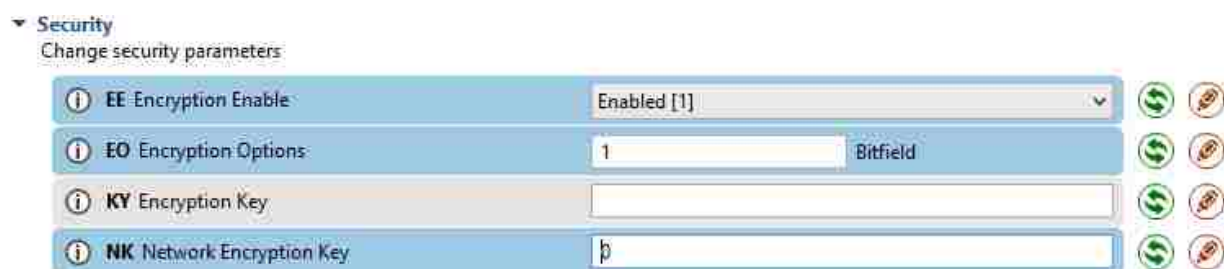


Figure 35: Coordinator security configuration for simulating ZigBee unencrypted network key attack.

In this simulation, we also configured our dummy device to use the “PAN ID” to be 1234. The “PAN ID” is crucial for joining the network, but since it’s easy to run a program to keep selecting all the “PAN ID” combinations and try each one of them to see which one is the correct one, we have decided to configure it ahead of time and focus on the unencrypted network key attack simulation. In terms of selecting the channel, we have set our dummy device to scan all of the 16 channels that ZigBee supports.

4.4.2 Attack Simulation

Unlike the physical attack simulation where we had to get hold the dummy ZigBee device in the network to perform our simulation, in this simulation of the unencrypted key attack we placed our dummy device within 400ft (the maximum range of our XBee Pro device) of the network location so we could be able to listen to the traffic and allow our dummy device to join it. After joining the network, a replay attack could be performed such as the one that was performed in the physical attack simulation, but in this simulation we chose to perform a DoS attack on the network to demonstrate a different attack behavior.

4.4.2.1 Attacker’s Node Joining the Network

We connected our dummy device to our PC and launched the XTCU application that detected our dummy device, and around 20 seconds later our dummy device was able to detect our home automation ZigBee network and join it without any problem. As we see in Figure 36, the ZigBee device at the top of the figure is our dummy device, and the other 2 remote modules are our coordinator and router for our home automated simulated network. Also we were able to detect the coordinator’s and the router’s MAC addresses; if our dummy device was not able to

join the network we would have not been able to see the two remote modules that were detected by our dummy device.

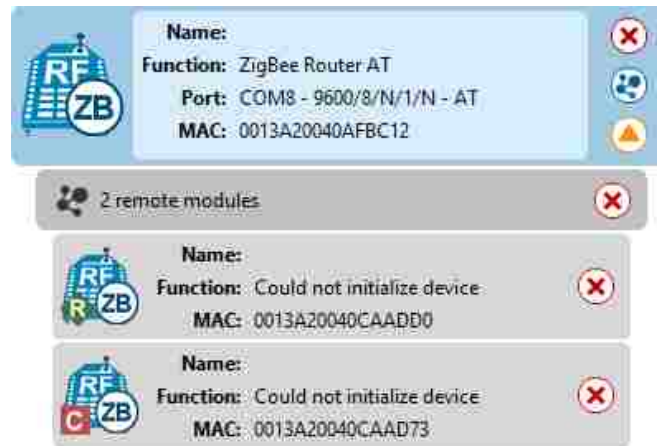


Figure 36: Attacker's Device Joins the ZigBee Network Setup with Unencrypted Network Key

Even though security was enabled, the ability of our dummy device to join the network means that the coordinator of the network was configured to send the network key unencrypted over air. On the other end, if we had disabled security in our dummy device it would have not been able to join our ZigBee secure network.

4.4.2.2 Perform DoS Attack

After joining the network using our dummy ZigBee device we have performed a Denial of Service (DoS) attack. Unlike the replay attack that we have performed in the attack simulation of the previous section, in this simulation we did not need to track the packages being sent, but we have injected a malicious packet multiple times in the network. The DoS attack was performed by flooding the network with dummy messages that caused the network to freeze and

denying the service thereafter. The dummy packet was injected in the network every 100ms for 2 minutes, and has caused the network to freeze for the whole period of the time that the attack has being performed; however, after we stopped the injection of the packet, the network went back to its normal behavior. Figure 37 shows the dummy packet containing the random data that has been created and the “Transmit Interval (ms):100”.

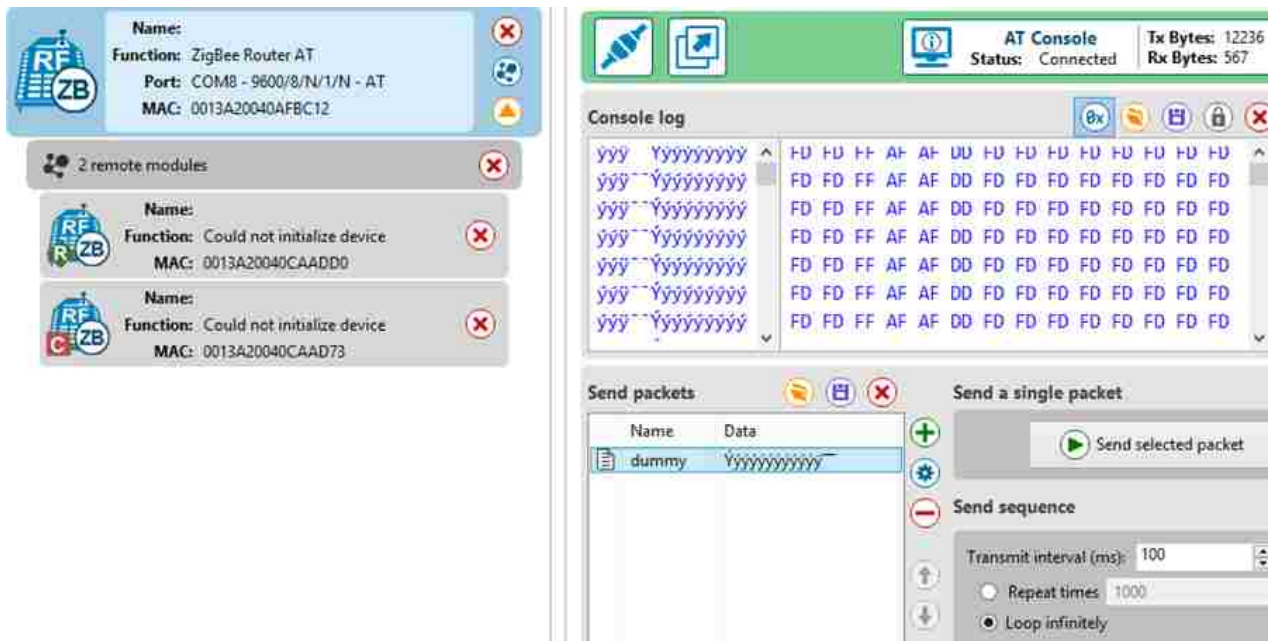


Figure 37: ZigBee DoS Simulation Attack

We have run another simulation where we have simulated the same attack for 10 minutes with the same interval of packet injections (100 ms) but this time the network did not recover and caused the network to crash. Both simulations are very critical in real world application; even though in the first simulation the network was able to recover after a short period of time, the consequences of such attacks can be unforgivable and could cause a life threat if these attacks were performed on devices that monitors medical patients.

Chapter 5: Secure Implementation of IoT Framework using ZigBee protocol

5.1 System Overview

This chapter discusses our own design and implementation of a secure IoT system that uses ZigBee protocol. This chapter goes over our own implementation of a secure IoT framework that uses ZigBee protocol. The security of our systems mainly relies on: configuring ZigBee devices in a secure manner using best practices learned from our attack simulations and other research papers and not using the default configuration; predicting potential malicious attack, by detecting the absence of node (device) that ZigBee protocol lacks and respond accordingly; adding another encryption layer to the data transmitted between the coordinator and the rest of the nodes (end devices or routers) that makes harder for an attacker to decrypt the data communicated; educating consumers about privacy and data security by giving them the autonomy to track in real time any motion activities detected around their house, and setup the time period that they should be notified in case of an attack or unexpected behavior.

Figure 38 shows our system's architectural components diagram that consists of:

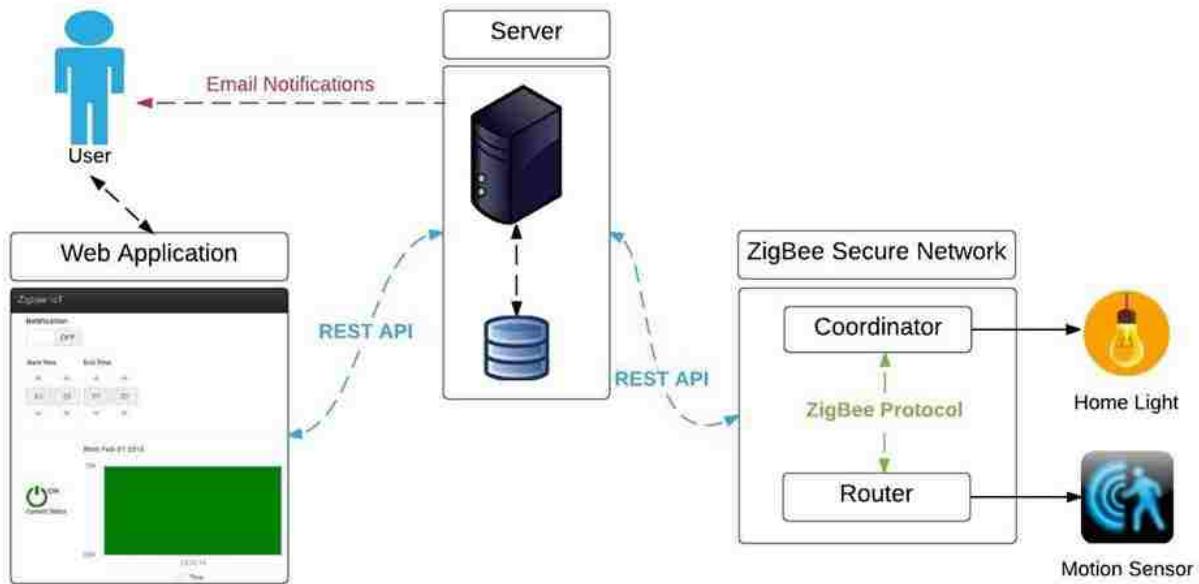


Figure 38: System's Component Diagram

- **A Router (sender):**

Is formed by an Arduino mega 2560 board that is connected to an infrared sensor, and to an XBee Pro S2 through an XBee module and configured as a ZigBee router using XTCU software (refer to section 4.2.1 Hardware for further features information). Once the infrared sensor detects a motion the Arduino board will process the data and send it serially to the XBee Pro S2 device which sends it over air to the receiver.

Figure 39 shows from left to right respectively the infrared sensor, the Arduino board, the XBee module, and the XB Pro S2. Figure 40 show the router assembled together.

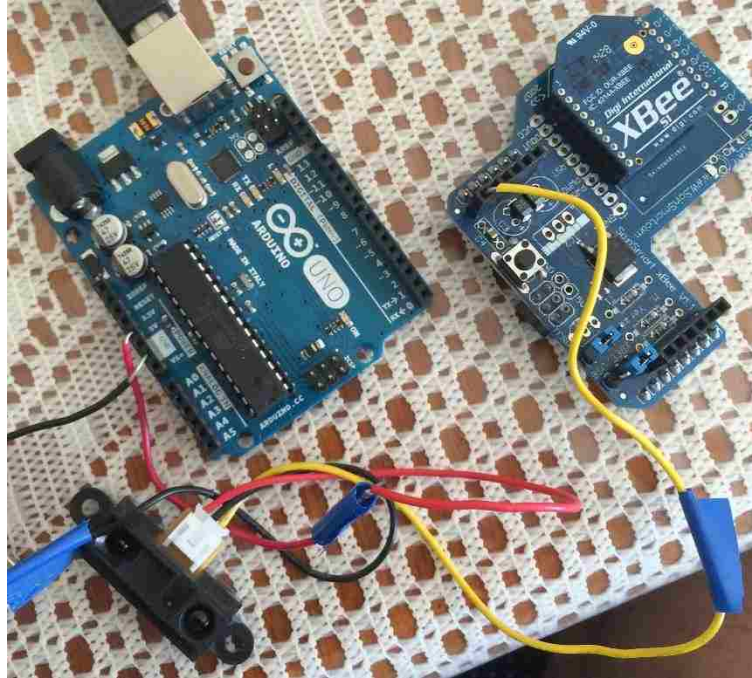


Figure 39: Router Parts and Connections

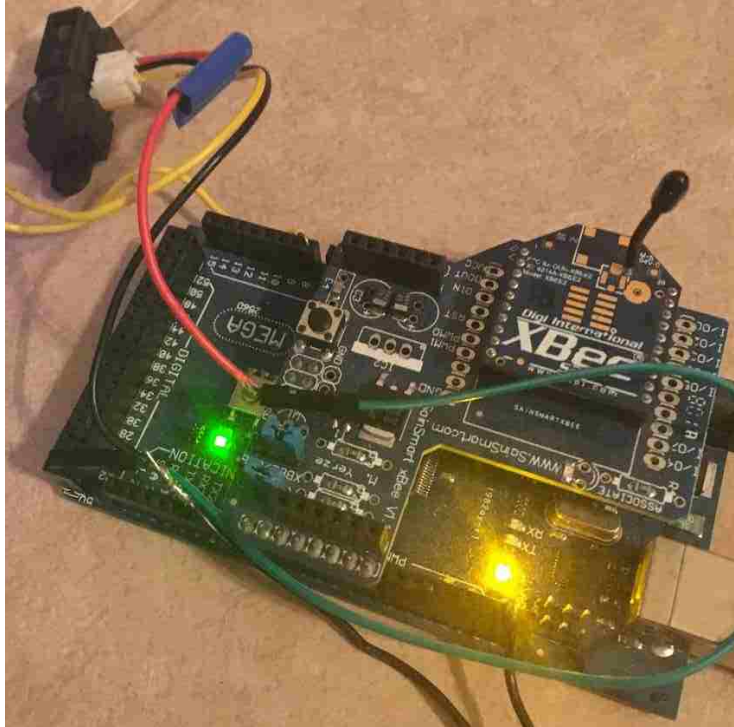


Figure 40: Router Assembled

- **A Coordinator (receiver):**

Is formed by Arduino mega 2560 board that is connected to an Ethernet shield and an XBee Pro S2 configured as a ZigBee coordinator. Once the XBee Pro S2 receives the data over air from the XBee router of the sender, it will send the data serially to the Arduino board which will process the data and notify the server through the Ethernet shield. The Ethernet shield functionality is to connect our coordinator to the internet which allows the coordinator to notify our server of any motion or suspicious attack through the REST API calls (refer to section 5.2.1 REST APIs).

Figure 41 and 42 are pictures of the coordinator taking from different angles; going from bottom to top, the 1st layer is the Arduino board connected to the 2nd layer the Ethernet shield which is

connected to the 3rd layer the XBee module that connects to the last layer the XBee Pro device. Also, as you see in Figure 40 the Ethernet shield is connected to an Ethernet cable that is plugged into our internet router.

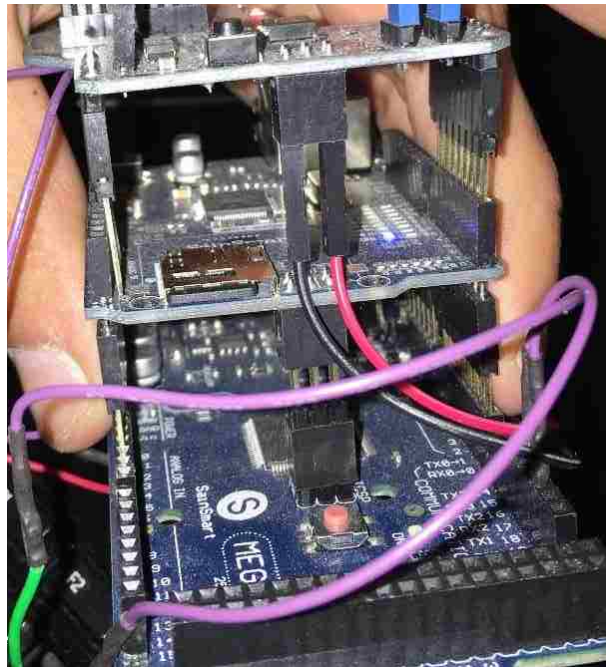


Figure 41: Coordinator Back View

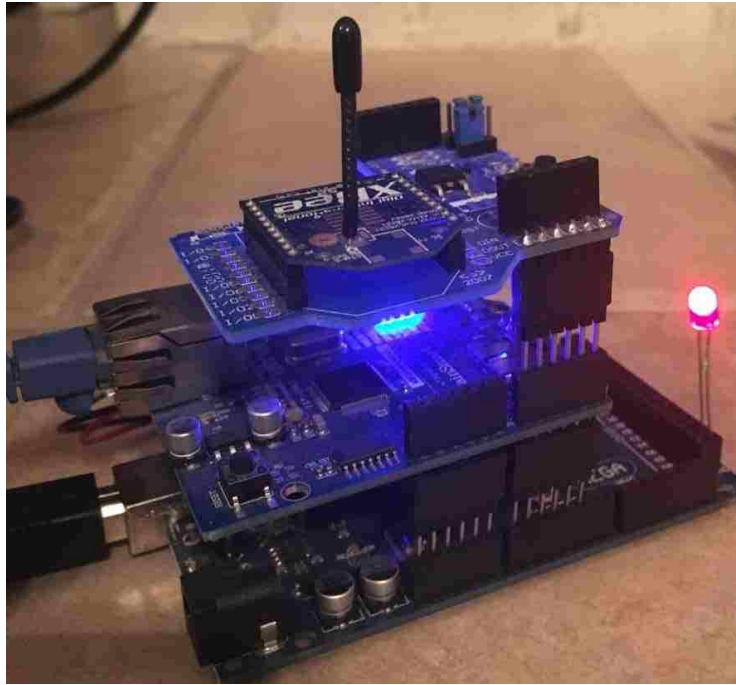


Figure 42: Coordinator Side View

- **Server:**

Implements the REST API methods to allow its communication among the web application and the coordinator. In addition, the server saves the data and retrieves it to from the mongoDB, and sends an email notification to the user upon suspicious activities such as removing a node from the network, tempering the normal flow of the system, and detecting motion activities in period of time that is not allowed by the user.

- **Web Application:**

Allows the user to track motion detection in real time, turn on/off notification, and set a period of time in which the system should sends him/her the notification. The time and date of

the motion detected is displayed by the web application and it is saved in the mongoDB to allow future features development related to data analytics.

5.2 Server

Our server is implemented using node.js; “Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient” [41]. Our node.js is installed using the node.js installer <https://nodejs.org/en/>, that also installs Node Package Manager (Npm). “Npm is the default package manager for the JavaScript runtime environment Node.js” [42]. Npm helps developers to find, share, and reuse packages(modules) of code that are located in the following online repository <https://www.npmjs.com/> [42]. Our server is comprised of the following Npm modules: Express.js is used as a web application framework; socket.io to enable real-time bidirectional event-based communication between server and client [43]; mongoose for querying, casting, writing business logic, and modeling mongoDB data; nodemailer for sending email notifications. For storing data, we have used mongoDB, a NoSql database that is built on an architecture of collections and documents [44] for its compatibility and ease of use with node.js. In addition to the Npm modules and the mongoDB, our server implements three Representational State Transfer Application Programming Interfaces (REST APIs) methods. These REST APIs will allow our XBee device (the coordinator) to save data into our mongoDB so it could be retrieved in real time by our client app. In our implementation we did not focus at the moment on securing or encrypting the REST APIs since most of the IoT systems implemented does that, and we will be adding security to the APIs in our future work.

5.2.1 REST APIs

5.2.1.1 Save Sensor state

Save the light status. Table 5 shows the Request API, and Table 6 shows its Response.

sensor-state: is set to “L” (low) if no motion is detected, or to “H” (high) if motion is detected.

Method	URL	Parameter	Value
POST	apis/dataApi/	[sensor-state]	String

Table 5: Server Rest API – Save Sensor State Request Method

Status	Response
200	{ “success”: “true” }
401	{ “error”: “Invalid data” }

Table 6: Server Rest API – Save Sensor State Response Method

5.2.1.2 Get most recent light state and its date and time

Get the latest light state and the date and time of its occurrence. Table 7 shows the Request API method, and Table 8 its Response.

Method	URL	Parameter	Value
GET	apis/dataApi/	-	-

Table 7: Server Rest API – Get most recent light state and its date and time Request Method

Status	Response
200	<pre>{ "data": [{ "status": "on", "dateRecieved": "Mon Nov 02 2015" }] }</pre>
500	<pre>{ "error": "latest light state could not be retrieved" }</pre>

Table 8: Server Rest API – Get most recent light state and its date and time Response

5.2.1.3 Email Notification

Notify user by email upon motion detection, or upon a possible malicious network attack. Table 9 shows the Request API method, and Table 10 shows its Response.

user-id: the unique id of each user using the system. s its Response.

Method	URL	Parameter	Value
POST	Notify/	[user-id]	String

Table 9: Server Rest API – Email Notification Request Method

Status	Response
200	{ "success": "true" }
500	{ "error": "notification could not be sent" }

Table 10: Server Rest API – Email Notification Response

5.3 Web Application

Our web application provides the user with the ability to monitor the light, and the motion detector status at any time and from anywhere. Also, it will allow the user to disable or enable notifications that are sent upon detection of any suspicious activities such as an object presence during unexpected times, or a potential system hack. Figure 43 shows a snap shot of our web app user interface (UI), where the notification can be turned on or off, the time interval of when the notification should be fired, the current status of the light (“ON” or “OFF”, depending if motion is detected or not) and a chart (graph) that will display the status of the light and the date and time of any recorded activity.

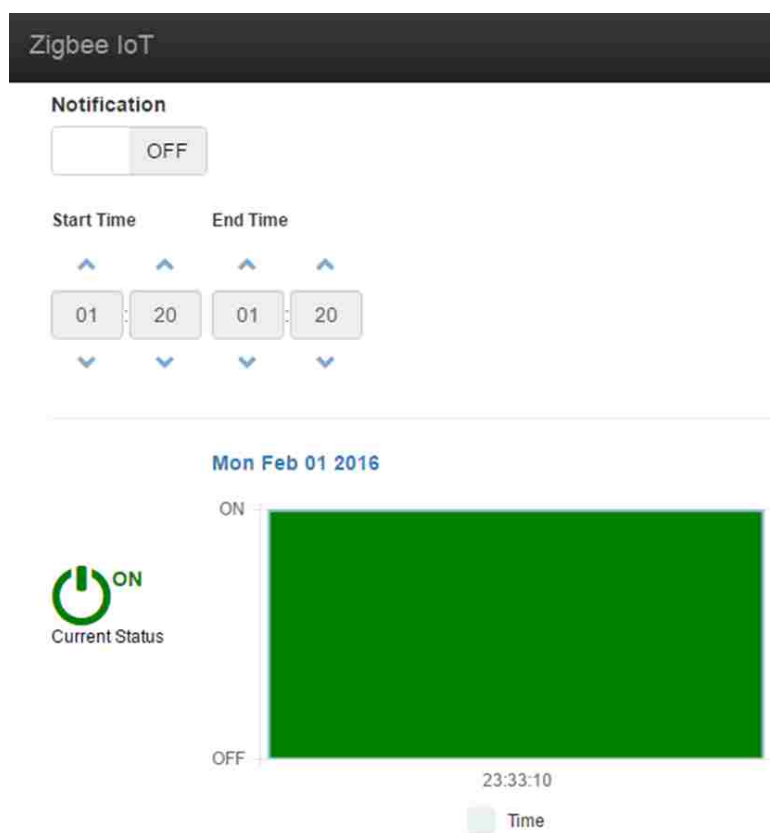


Figure 43: Web Application UI

5.3.1 AngularJS and the Mode-View-Controller Structure

Our web application is built using the open source AngularJS framework. “AngularJS is a very powerful JavaScript Framework. It is used in Single Page Application (SPA) projects. It extends HTML DOM with additional attributes and makes it more responsive to user actions” [45].

The best practice to structure the code in an AngularJS app is to use the Model-View-Controller (MVC) design pattern to decouple the code structure and to separate concerns. AngularJS, however, does not implement the MVC in a traditional sense, but rather, it uses something closer to MVVM (Model-View-ViewModel) pattern; AngularJS team refers to it humorously as Model View Whatever. Nonetheless, AngularJS is still considered as an MVC based framework. The MVC framework is based on three parts as shown in Figure 44:

- **Model:**

The model is responsible for maintaining application data. It receives requests from the view and the controller to update itself.

- **View:**

The view is responsible for displaying the data upon a trigger by the controller.

- **Controller:**

The controller is responsible for the application logic and controlling the interaction between the Model and the View. The controller receives requests from the user and responds by performing business operations that could modify the state of the data model and the view.

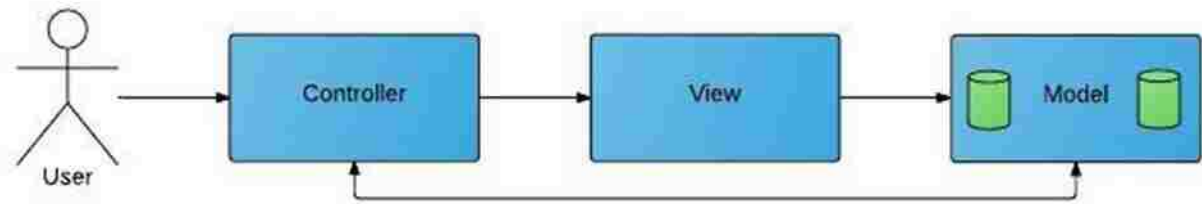


Figure 44: Model View Controller

AngularJS core features allow developers to implement the MVC architecture and single application concept in a very simple and clean way. Some of the main features of AngularJS that allows such implementation are:

- **Data-biding:**

Allows automatic synchronization of data between model and view.

- **Scope:**

Are the objects that refer to the model in the MVC architecture.

- **Controllers:**

Are the controllers that refer to the MVC architecture and are bound to the view and the scope (model).

- **Templates:**

Are the rendered views (in MVC) and displays the information from the scope (model) and the controller. The templates could be a single file e.g. index.html, or multiples view in on page and are called partials.

- **Directives:**

Are used to create custom HTML tags, attributes, css and more.

- **Routing:**

Is used to switch views in single application concept.

- **Services:**

Are singleton objects and many of them come built-in with AngularJS. For example, \$http service that creates XMLHttpRequests.

5.3.2 Implementation/Design

Our web application design is based on the architecture of AngularJS framework that uses the Model-View-Controller pattern. Our app consists of the following components:

A “Main-View” (called Main-Template in AngularJS terminology), a “Main-Controller”, a “Model”, a “SensorStatus-Controller”, and a “SensorStatus-View” (called Partial-Template in AngularJS terminology). The responsibilities of each component are:

- **Main-View:**

Is responsible for displaying the data from the Main-Controller and the “Model”.

- **Main-Controller:**

Is responsible for some the application logic and controlling the interaction between the Model, the Main-View, and the Sensor-Status-Controller (Partial-Controller).

- **Model:**

Is responsible for maintaining the application data and it is instantiated within the Main-Controller. In AngularJS the Model is referred to “Scope”. For example, to save the sensor data in an AngularJS application, a developer will use the following statement in the controller “\$scope.sensorData = “ON”“.

- **SensorStatus-Controller:**

Is responsible for the rest of the application logic that is not handled by the Main-Controller, and the interaction between the Main-Controller, the SensorStatus-View (Partial-Template), and the Model.

- **SensorStatus-View:**

Is responsible for displaying the data from the SensorStatus Controller.

In the sub-sections below we will discuss our web app flows, and the interactions among the app’s components using sequence diagrams.

5.3.2.1 Update Sensor Status and Notification Flow:

The sequence diagram in Figure 45 goes over the flow of updating the light status (“ON” meaning motion detected, and “OFF” no motion detected) and notifying the user by email in case

of motion is detected. Upon changes of light status, the SensorStatus-Controller will receive a notification from the server using the socket.io method “io.socker.emit(sensorStatus)” [23], then the SensorStatus-Controller will call the Main-Controller “upDateModel()” method in order to update the model with the new the sensor data. Once, the sensor data has been updated, the Main-View will receive a notification and update its view, by displaying the current status of the light as “ON” or “OFF”. Also the SensorSatus-Controller will notify its own partial view/template (SensorStatus-View) to update its chart to display the status of the light, and the date and time of the occurrence. Lastly, the SensorStatus-Controller, will check if the notification is turned on by the user, by calling the “checkNotification()” method of the Main-Controller which checks the Model data. If the notification is off, the flow ends and no notification is sent to the user; otherwise, the SensorStatus-Controller will call “sendEmailNotifcation(user-id)“ method in the App-Service that will make a REST call “\$http.post(url/notify/user-id)” to the server which will send an email to user.

UPDATE SENSOR STATUS & NOTIFICATION FLOW - SEQUENCE DIAGRAM

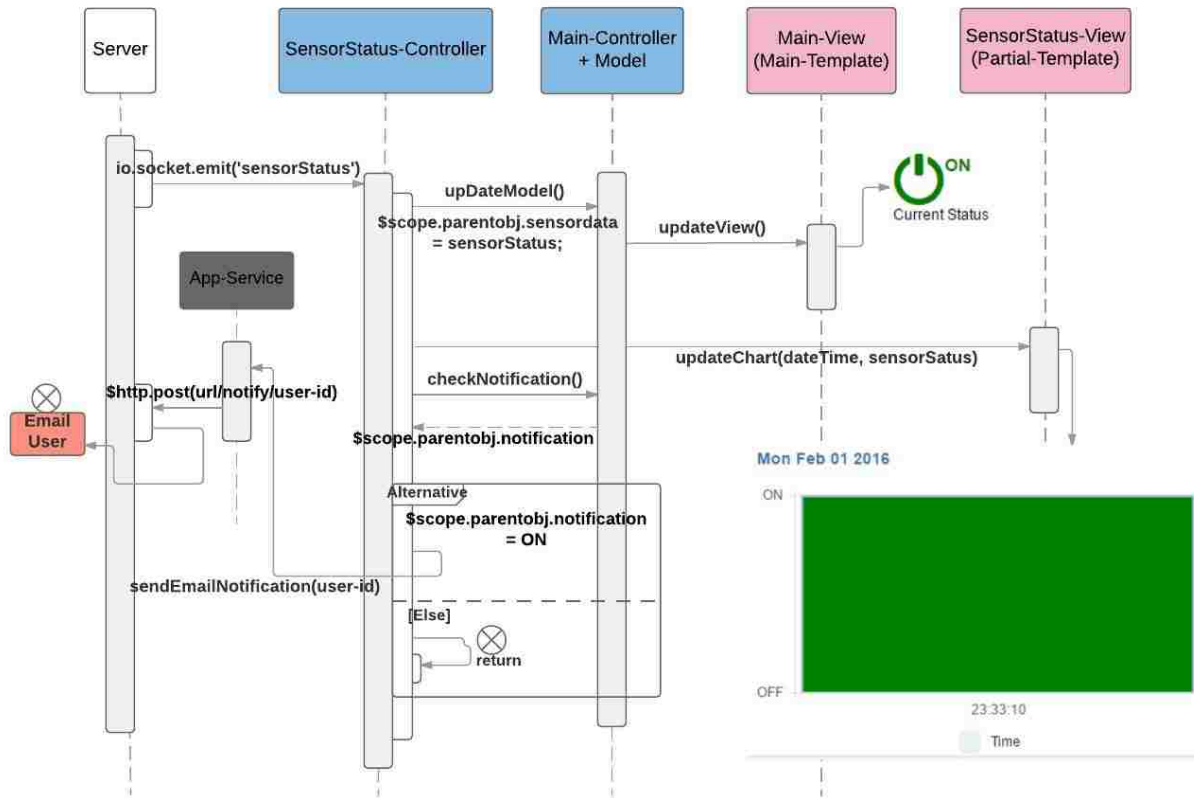


Figure 45: Update Sensor Status & Notification Flow - Sequence Diagram

5.3.2.2 Enable Notification Flow

The sequence diagram in Figure 46 goes over the flow of enabling the notification by the user.

ENABLE NOTIFICATION - SEQUENCE DIAGRAM

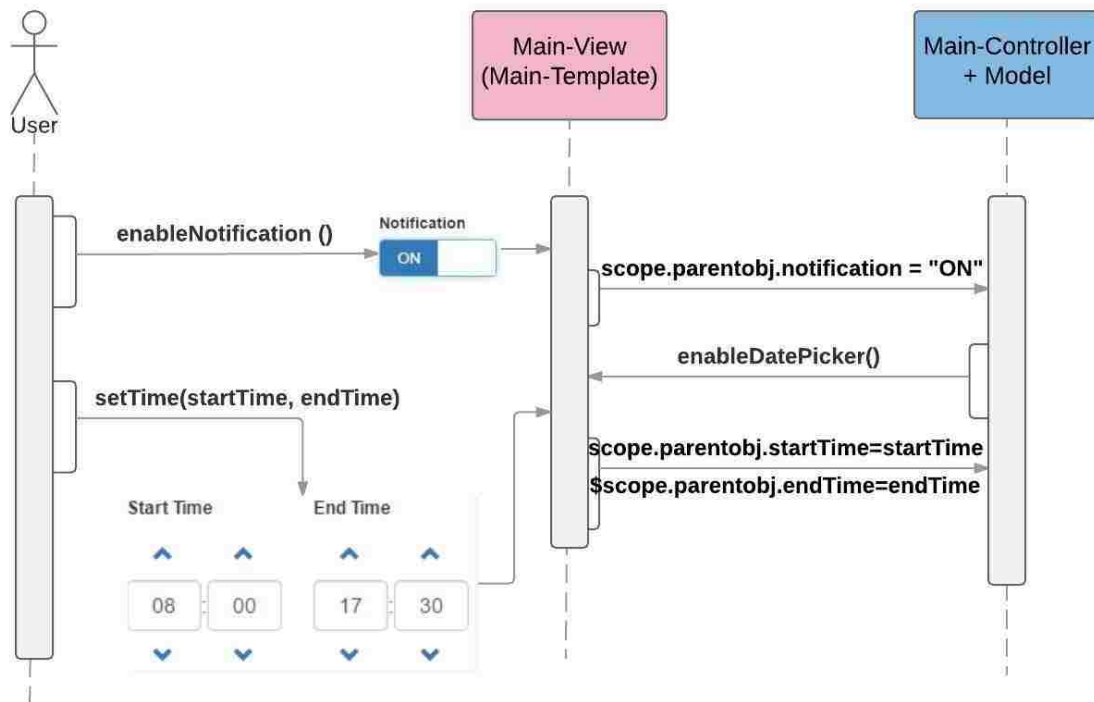


Figure 46: Enable Notification - Sequence Diagram

The user will enable the notification by switching the notification button in the Main-View to “ON”. The Main-View will set the notification variable “scope.parenobj.notification=ON” in the Model (scope) which will trigger the Main-Controller to enable the timer dialog in the Main-View by calling “enableDatePicker()” and allowing the user to set the time interval of when he should be notified by setting the start and the end time period. Once the timer dialog is enabled, the user will be able to set the time interval and the Main-View will update the Model by setting the start and the end time variables “scope.parentobj.startTime” and “scope.parentobj.endTime”.

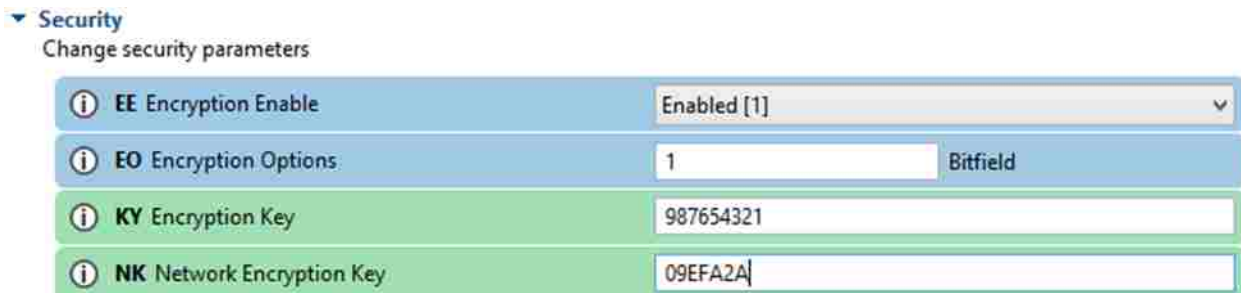
The user will be notified by email every day in the interval of time specified in case of any motion detected, or in case of a potential attack on the system.

5.4 IoT System Secure ZigBee Configuration

We have designed our IoT system to use the best practices of securing a ZigBee network. Securing our ZigBee network heavily depends on configuring our devices in a secure manner. The following steps describes the configuration used in our ZigBee network to prevent any security breaches that could make our network vulnerable to the attacks described in Chapter 4: Attacks Simulations on ZigBee, and to prevent any unidentified ones.

- **Prevent the acquisition of ZigBee keys by an attacker**

In order to prevent such attack, we have preloaded the link key and the network key at the time of the configuration of our ZigBee coordinator and router. Figure 47 shows the configuration of the coordinator where we have enabled security by enabling encryption (EE parameter), we have used the coordinator as trust center by setting the “EO” parameter to ‘1’ to not allow the network security key to be sent unencrypted over air, we have configured/preloaded the linked key (KY parameter) to be “987654321”, and the network encryption key (NK parameter) to be “09EFA2A”. In this case the network key(NK) will be sent encrypted by the linked key to the router willing to join the network.



▼ Security
Change security parameters

EE Encryption Enable	Enabled [1]
EO Encryption Options	1 Bitfield
KY Encryption Key	987654321
NK Network Encryption Key	09EFA2A

Figure 47: Coordinator Security Configuration

Besides the coordinator's configuration, we also configured the router with the same EE, EO, and KY parameters as the coordinator. Figure 48 shows the configuration of the router, and as you noticed that the router does not have a network key (NK) configured since it will be sent encrypted by the coordinator with the link key (KY)

By setting the configuration keys, we have also prevented the default key configuration that these devices are pre-loaded with.

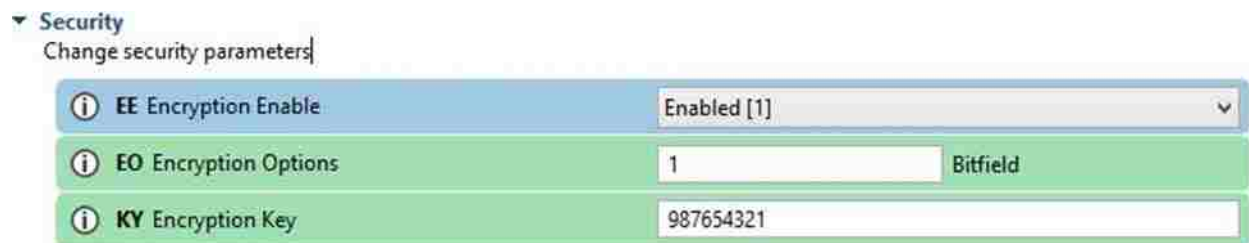


Figure 48: Router Security Configuration

- **Prevent Physical attack:**

To prevent any potential physical attack, and besides placing our router in a safe place, we have implemented a “heart beat” between the coordinator and the router that will notify the user in case the coordinator does not receive any signal from the router for a time-lapse of 2 seconds. The” heart beat” implementation is added to cover the lack of detection of missing nodes in the network by the ZigBee protocol.

In addition to these configuration, we have also set the PAN ID of the network in the coordinator and the router, and have specified the channel mask that the network should operate on.

5.5 Network Functionality and Implementation

Our attacks simulations on a ZigBee network have shown that securing the ZigBee network by only securing devices configurations is not sufficient. For example, removing a node from the ZigBee network is not detected by the network and specifically by the coordinator, and does not generate and send a new network key to the other devices that are still in network, thus allowing the attacker to compromise the ZigBee network in various ways (refer to Chapter 3 and Chapter 4). Detecting the absence of a node in the network is crucial to prevent any stolen ZigBee device to be reused thus to re-join and compromise the network. To solve this problem, we have implemented a “heart beat” between the coordinator and the router. The “heart beat” is an encrypted message sent by the sender repeatedly every 200ms to indicate its presence to the receiver; in case the receiver does not receive any message in the period of 2 seconds it will notify the user. Implementing the “hear beat” will not only warn the user about a possible malfunctioning of the sender but also about its nonexistence in the network and will prevent any possible future network attacks.

In addition to the “heart beat” implementation, and the secure configuration of the ZigBee devices, we have also encrypted all the data that being transmitted at the application layers. For example, the sensor state data and the “heart beat” data are also encrypted by the sender and decrypted by the receiver using the AESLib.h library. The AESLib.h library uses 128 bit or 256-bit encryption key. The encryption at the application layer and the encryption at the network layer adds more security to our system, thus reducing the chances and the ability of an attacker to decrypt the data.

Figure 49, and 50 shows different snapshots of our IoT framework in real time. Our web application on left, and on the right the messages received by the coordinator from the router in real time. The coordinator receives encrypted messages from the router and decrypts them to get the data such as “High”, “LoW”, or “Beat”. The encrypted data and their decryption are made available by connecting the coordinator serially to the PC and opening the serial port console window from the Arduino IDE that was also used to program the Arduino boards (coordinator and router).

Figure 49 shows on the left the initial state of the web application where we don't see any activity in the timeline, and on the right the signal of the “Beat” received by the coordinator signaling the presence of the router only; meaning no motion detection, and no suspicious unexpected behavior.

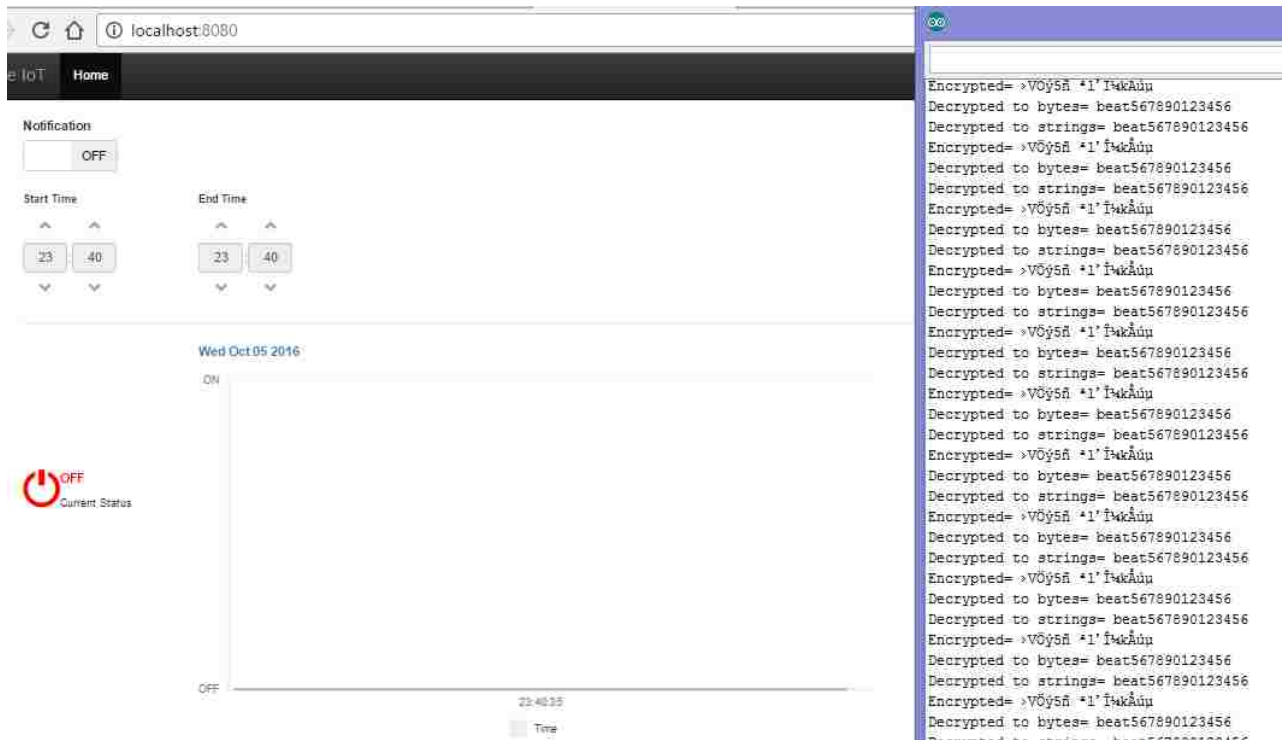


Figure 49: IoT System Real Time Snapshot – No Motion Detected

Figure 50 shows the detection of motion, where the web application is updated in real time and showing that the light is on, and on the right the signal “H” received by the coordinator signaling the presence of an object.

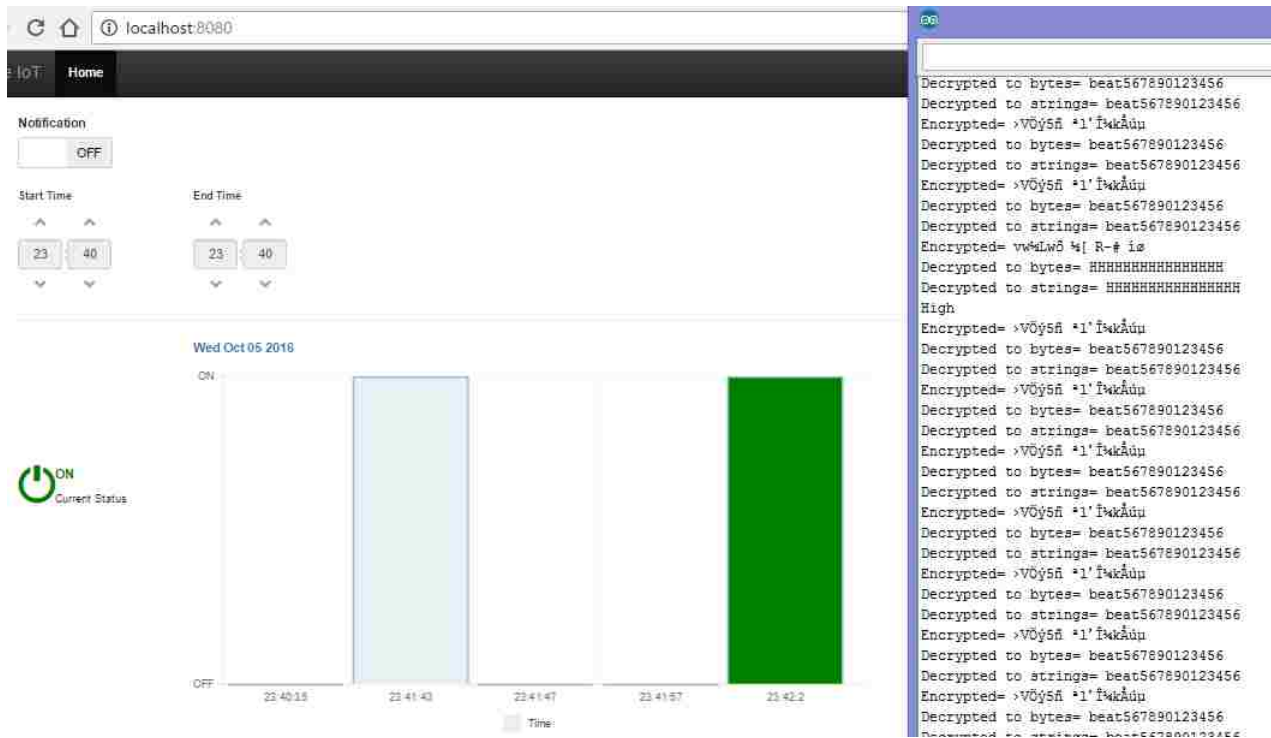


Figure 50: IoT System Real Time Snapshot – Motion Detected

5.5.1 Network Functionality Use Cases

The following section goes over our system’s secure implementation and use cases scenarios presented in the form of sequence diagrams. (Refer to section 4.2.1 REST APIs, for further details related to the REST calls).

5.5.1.1 Notify User in Case of Invalid “Heart Beat” Signal

To confirm its presences in the network, the sender will send an encrypted “heart beat” signal to the receiver every 200 ms. The receiver in its turn will decrypt the “heart beat” message and validates it; if the “heart beat” message is valid the flow ends without any actions, otherwise

the receiver will make an “HttpRequest” (REST API call) to the server that will notify the user by email about a suspicious attack, as shown in Figure 51.

NOTIFY USER IN CASE OF INVALID HEART BEAT MESSAGE - SEQUENCE DIAGRAM

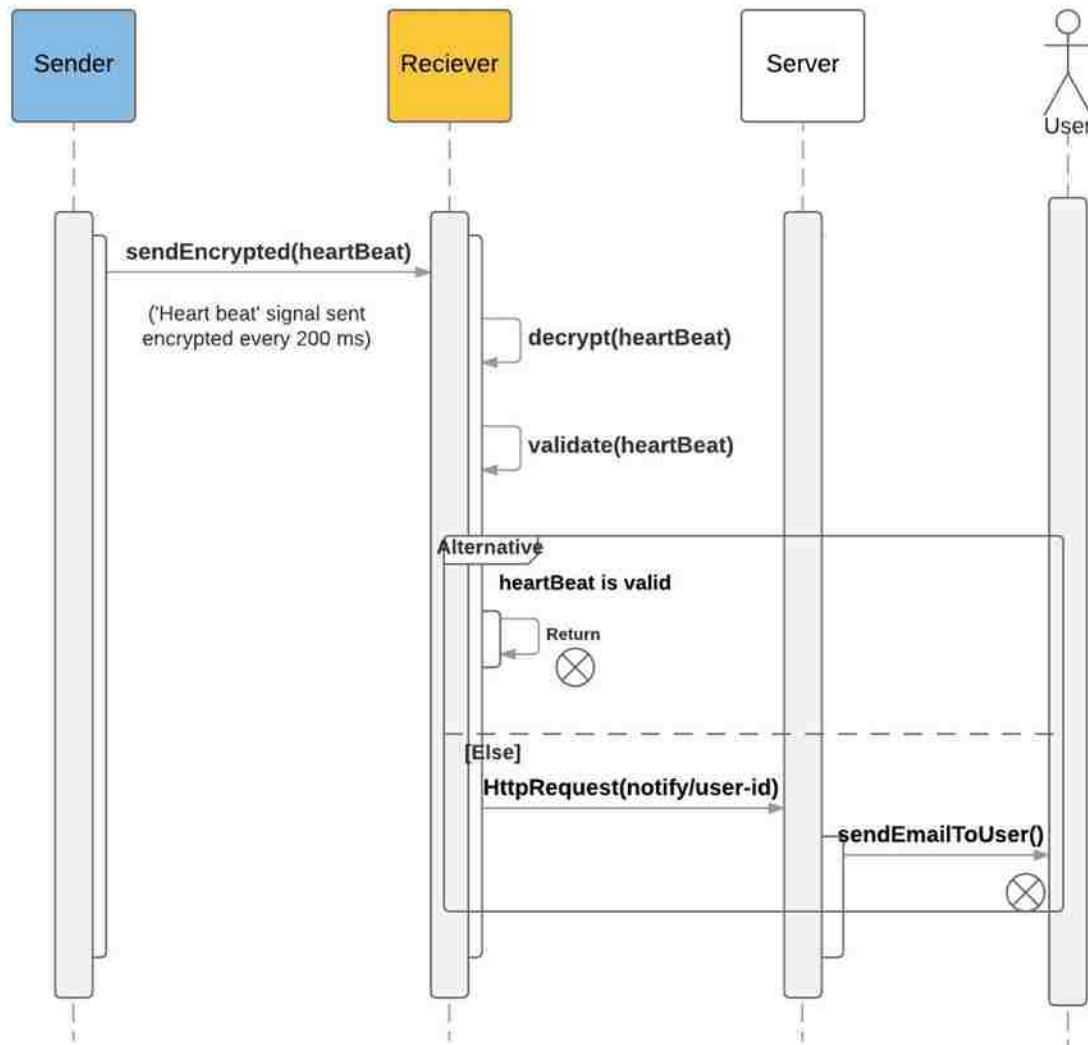


Figure 51: Notify User in Case of Invalid Heart Beat Message – Sequence Diagram

5.5.1.2 Notify User in Case of Invalid "Heart Beat" Signal

Another scenario where the user will get notified by our system, is when the sender becomes unavailable. In case the receiver does not receive any "heart beat" signal from the sender within 2 seconds the receiver will make an "HttpRequest" (REST API call) to the server that will notify the user by email, as shown in Figure 52.

NOTIFY USER IN CASE OF HEART BEAT MESSAGE NOT RECIEVED - SEQUENCE DIAGRAM

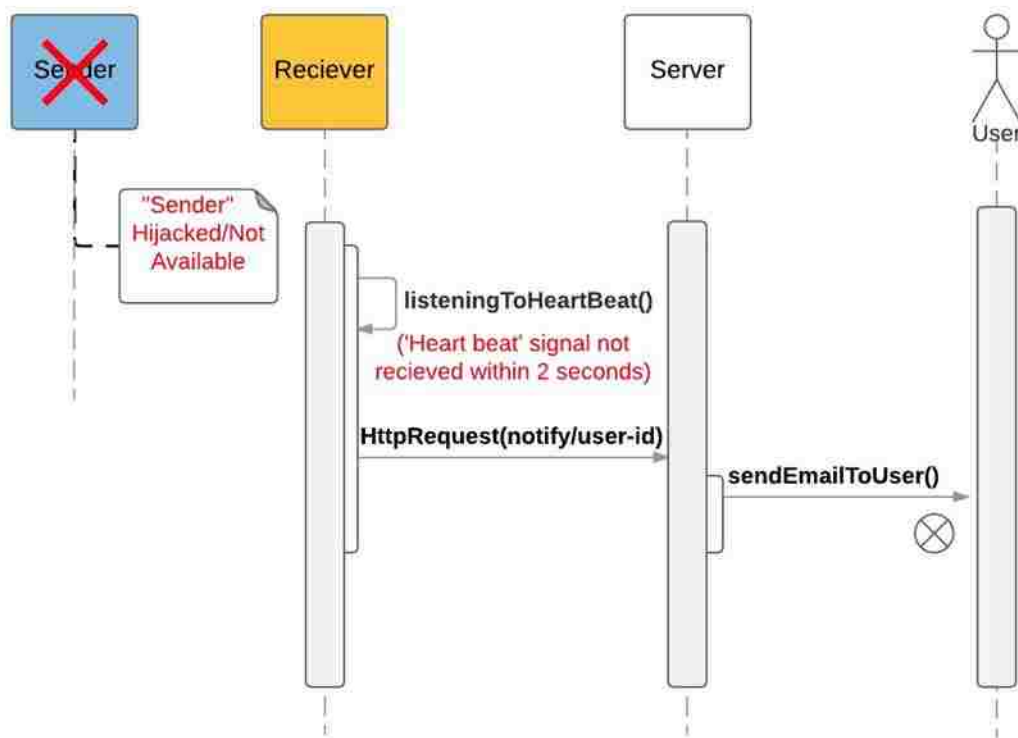


Figure 52: Notify User in Case of Invalid Heart Beat Message – Sequence Diagram

5.5.1.3 Notify User in Case of Invalid Sensor Data, and Update the Client in Case of Valid Ones

Upon receiving a new data from the motion sensor, the sender will read the sensor state, then compare it to the previous most recent one; if the new state is equal to the previous state, the flow ends, otherwise the sender encrypts the new state and send it to the receiver. The receiver upon receiving the new sensor state decrypts it and then validates it; if the state is valid, the receive will call the server's REST API "apis/dataApi/sensor-state" using an "HttpRequest" and the server in its turn notifies the app to update its states; if the state is invalid the receiver will call the server's REST API "notify/user-id" using an "HttpRequest" and the server will notify the user about a suspicious activity or potential attack, as shown in Figure 53.

NOTIFY THE USER IN CASE OF INVALID SENSOR DATA, AND UPDATE THE CLIENT APP IN CASE OF VALID ONES -
SEQUENCE DIAGRAM

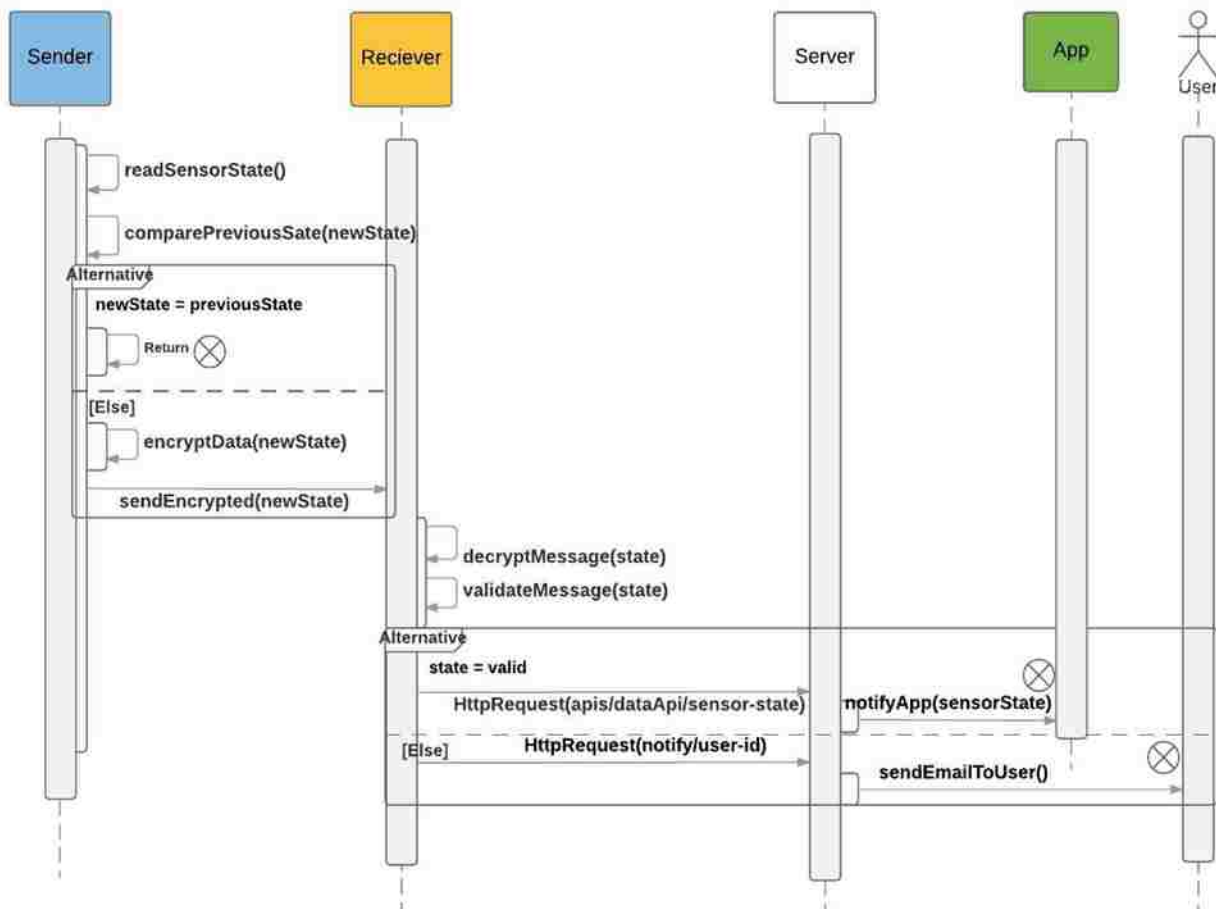


Figure 53: Notify the User in Case of Invalid Sensor Data, and Update the Client App in Case of Valid ones – Sequence Diagram

5.5.2 Router’s Implementation/ Source Code

Figure 54 shows the router’s source code where the router (Arduino board) is connected to the Infra-Red (IR) sensor using digital pin 8, and the state of the sensor is initialized as “L” (low). The router will read the sensor state every 0.5 seconds, then converts the voltage reading

to an actual distance that indicates the presence or the absence of an object. The router, will send an encrypted “heart beat” signal to the coordinator every 0.2 seconds indicating its presence. Every time a new sensor data is collected, the router will compare it to the previous sensor data (state), and if the 2 states are not equal it will encrypt the new sensor state and send it to the coordinator indicating the change.

The “setup ()” sets up the serial communication between the Arduino board and the XBee.

The “loop ()” method encrypts the “heart beat” signal and the sensor state, and sends it to the XBee which sends it over air to the coordinator.

Router's Source Code

```
#include <AESLib.h>
int sensorPin = 8; // Infra-Red (IR) sensor connected to digital pin 8
String state = "L";

void setup() {
  Serial.begin(9600);
}

void loop() {
  float voltage = analogRead(sensorPin)*0.005;
  float distance = 65*pow (voltage, -1.10); //convert the IR sensor voltage to
distance

  uint8_t key [] = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15}; //encryption key
  delay (100); //100ms delay
  char data [] = "beat567890123456"; // heart beat data signal
  aes128_enc_single (key, data); // encrypt the data to be sent
  Serial.println (data); // send data serially to Xbee
  Delay (200);

  if (distance<=150 && distance>10) { // check if motion detection
    if (state=="L") { // if previous state is low
      char data [] = "HHHHHHHHHHHHHHHHHH";
      aes128_enc_single (key, data); //encrypt "high" state data
      state="H";
      Serial.println(data); // send "high" state to coordinator
      Delay (200);
    }
  } else { // if no motion is detected
    if (state=="H") { // if previous state is high
      char data [] = "LLLLLLLLLLLLLLLLLLLL";
      aes128_enc_single (key, data); //encrypt "low" state data
      state="L";
      Serial.println (data); //send "low" state to coordinator
      Delay (200);
    }
  }
}
```

Figure 54: Framework Router's Source Code

5.5.3 Coordinator's Implementation/ Source Code

Figure 55 shows the coordinator's source code where the coordinator (Arduino board) is connected to an Ethernet shield which is connected to the user's router to allow the coordinator to send data to the web application server. The coordinator will configure the Ethernet shield by setting its MAC, and IP address. The coordinator is also connected to an LED acting as the light that turns on and off depending on the data send from the router (motion detected or not). Upon receiving any data from the router the coordinator will first decrypt it using the same key set in the router, validates it, and execute its action accordingly.

The "setup ()" sets up the serial communication between the Arduino board and the Xbee, configures the Ethernet shield, sets up the time interval "t.setInterval (setTime, getData)" for when to call the web application server APIs, and configures the digital pin that is connected to the LED.

The "loop ()" reads the data received from the router, decrypts it, validates it, keeps track of the last time it has received the "heart beat" signal, and calls the "t.run()" that sends the data received to the web server application.

The "restConnect (IPAddress getServerAdd)" method calls the web application server API's, for example: "notify/[user-id]" to notify the user about a suspicious activity, or "/apis/dataApi/[sensor-state]" to send the current sensor state.

Coordinator's Source Code

```
#include <Ethernet.h>
#include <SimpleTimer.h>
#include <AESLib.h>
#include <SPI.h>

int ledPin = 53;
byte MAC[] = {0xFE, 0xAB, 0xAA, 0xBB, 0xCC, 0xFD};
IPAddress IP(192,168,1,101);
IPAddress server(192,168,1,102);

EthernetClient getClient;
int setTime;
SimpleTimer timer;
String dataSend = "";

long currentMillis = 0;
long previousMillis;
long interval = 2000;
long result;

void setup() {
  Serial.begin(9600);
  while(!Serial) { ; }
  if (Ethernet.begin(mac) == 0) {
    Ethernet.begin(MAC,IP);
  }

  delay(2000);
  setTime = 500;
  timer.setInterval(setTime, getData);
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, LOW);
} //end of setup()

void loop() {
  uint8_t key[] = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};
  String content = "" ; char character; char charBuf[50];
  previousMillis = millis();
  result = previousMillis - currentMillis;
  Serial.print("result = ");
  Serial.println(result);

  while(Serial.available()) {
    character = Serial.read();
    content.concat(character);
    delay (10);
  }
}
```

```

if (content.length() == 16) {
  Serial.print("Encrypted = ");
  Serial.println(content);
  dataSend= content;
  delay(10);

  dataSend.toCharArray(charBuf, 50);
  aes128_dec_single(key, charBuf);

  Serial.print("Decrypted to bytes= ");
  Serial.println(charBuf);

  dataSend = charBuf;
  Serial.print("Decrypted to strings= ");
  Serial.println(dataSend);

  if (dataSend == "HHHHHHHHHHHHHHHHHH") {
    Serial.println("High");
    digitalWrite(ledPin, HIGH);
  }

  if (dataSend == "LLLLLLLLLLLLLLLLLL") {
    Serial.println("LOW");
    digitalWrite(ledPin, LOW);
  }

  if (dataSend == "beat567890123456") {
    currentMillis = millis();
    Serial.println("beat567890123456");
  } else { dataSend = invalid; }

  delay(10);
}
}
t.run();
} // end of loop()

void restConnect (IPAddress getServerAdd)
{
  if ((result > interval && dataSend == "beat567890123456") || dataSend ==
invalid) {
    Serial.println('in');
    if (getClient.connect(getServerAdd, 8080)) {
      getClient.println("POST /notify/112 HTTP/1.1");
      dataSend="";
      getClient.stop();
    } else { Serial.println("failed to connect"); }
  }
}

```

```

} else {
    if (dataSend == "HHHHHHHHHHHHHHHH" || dataSend== "LLLLLLLLLLLLLLLLLLLL") {
        if (getClient.connect(getServerAdd, 8080)) {
            getClient.println("GET /apis/dataApi/" + dataSend + " HTTP/1.1");
            dataSend="";
            getClient.println("HOST: 192.168.1.102");
            getClient.stop();
        } else { Serial.println("failed to connect"); }
    }
}
}
} // end of restConnect()

void getData() {
    if (getClient.available()) {
        char c = getClient.read();
        Serial.print(c);
    }
    restConnect(server);
} //end of getData()

```

Figure 55: Framework Coordinator's Source Code

Chapter 6: Conclusion and Future Work

6.1 Conclusion

The importance of security in IoT and the domination of ZigBee protocol in home automation were the leading factors in writing this thesis. In this thesis we have surveyed ZigBee vulnerabilities, and shared some recent IoT real world attacks that ZigBee could also be a victim of. We have also performed first hand attacks simulations experiments of some of the most common attacks That have enabled us to design and implement an IoT framework that would prevent them. The IoT framework developed is not unique in its functionality, but its importance is in solving and suggesting security measures that should be implemented by any company that manufacture IoT products in general and that uses ZigBee protocol.

The developed IoT system based on the ZigBee protocol implements multiple layers of defense, secures all the layers and components of the ZigBee network, and predicts potential malicious attacks. The system would solve the problem of failing to detect a missing node in the ZigBee protocol by keeping a communication signal between any pair of communicating nodes in the network. Also, the framework developed adds another security layer between ZigBee communicating nodes, where we have encrypted the messages transmitted with an AES 128-bit encryption library. The framework implements a secure ZigBee device configuration, and negates the use of the manufacturers default configuration. In addition, we have highlighted the importance of educating users about security by giving them the autonomy to track in real time any motion activities detected around their house, and to setup the time period that they should be notified in, in the event of any suspicious activity.

6.2 Future Work

The work that have been done in this thesis solves many problems related to ZigBee and to IoT security vulnerabilities, however, more work and implementation is still needed to obtain an ideal secure IoT framework. Our future work, and as of the continuation of better securing IoT systems will revolve around:

- 1- Securing and encrypting the Rest Apis used in communication between the coordinator, the server and the web application.
- 2- Being prepared for an attack [17] [18] 46] by setting up an incident response process, and system recovery.
- 3- Considering data minimization by not collecting more data then needed.
- 4- Developing a mobile app that send notification to the user in case of potential attacks, or activity detection.
- 5- Securing the Interoperability of IoT systems and devices.

References

- [1] J. Gubby, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions", 2012
- [2] Gartner, Inc. "Gartner Says by 2020, a Quarter Billion Connected Vehicles Will Enable New In-Vehicle Services and Automated Driving Capabilities". Retrieved December 15th, 2015 from <http://www.gartner.com/newsroom/id/2970017>
- [3] R. Milman. "Bluetooth and ZigBee to dominate wireless IoT connectivity," Internet of Business: Apr. 29, 2016. Retrieved Sep 17th, 2016 from <https://internetofbusiness.com/iot-driving-wireless-connectivity/>
- [4] "Research and Markets has announced the addition of the "Global ZigBee Home Automation Market 2016-2020" report to their offering," TheStreet: Sep 22nd, 2016. Retrieved Oct 10th, 2016 from <https://www.thestreet.com/story/13749377/1/research-and-markets--global-zigbee-home-automation-market-drivers-trends-challenges-2016-2020--development-of-zigbee-standards-to-support-iot-high-initial-cost-of-smart-products-emergence-of-internet-connected-lightbulbs.html>
- [5] Symantec. "ISTR 20: Internet Security Threat Report". April 2015. Retrieved May 11th, 2016 from https://www.symantec.com/content/en/us/enterprise/other_resources/21347933_GA_RPT-internet-security-threat-report-volume-20-2015.pdf

- [6] E. Perez, S. Prokupez, and T. Cohen. CNN. “More than 90 people nabbed in global hacker crackdown”. 2014. Retrieved May 11th, 2016 from <http://www.cnn.com/2014/05/19/justice/us-global-hacker-crackdown/>
- [7] B. Montgomery. “The Internet of Secure Things - Identity Based Encryption 3.0”. Sep 13, 2015. Retrieved October 10th, from <https://www.linkedin.com/pulse/10-most-terrifying-iot-security-breaches-so-far-you-arent-montgomery>
- [8] T. Zillner. “ZigBee Exploited: The Good, The Bad and The Ugly”, in Black Hat (2015).
- [9] P. Radmand, M. Domingo, J. Singh, J. Arnedo, A. Talevski, S. Petersen, and S. Carlsen. “ZigBee/ZigBee PRO security assessment based on compromised cryptographic keys”. Digital Ecosystem and Business Intelligence Institute, Curtin University of Technology, Perth, Australia
- [10] O. Olawumi, K. Haataja, M. Asikainen, N. Vidgren, and P. Toivanen “Three Practical Attacks Against ZigBee Security: Attack Scenario Definitions, Practical Experiments, Countermeasures, and Lessons Learned”, in IEEE 14th International Conference on Hybrid Intelligent Systems (HIS2014), At Kuwait. DOI: 10.1109/HIS.2014.7086198
- [11] P. Egli, “Susceptibility of Wireless Devices to Denial of Service Attacks”, Technical white paper, Netmodule AG, 2006
- [12] J. Brodsky and Anthony McConnell, “Jamming and Interference Induced Denial-of-Service Attacks on IEEE 802.15.4-based Wireless Networks”, Tech. Rep., Digital Bond’s SCADA Security Scientific Symposium, 2009
- [13] J. Wright. “KillerBee: Practical ZigBee Exploitation Framework”, in ToorCon, 2009

- [14] CISCO.” Securing the Internet of Things: A Proposed Framework”. Retrieved Aug16th, 2016 from <http://www.cisco.com/c/en/us/about/security-center/secure-iot-proposed-framework.html>
- [15] I. B.Pasquier, A. A. El Kalam1, A. A. Ouahman, and M. De Montfort. “A Security Framework for Internet of Things”. Springer International Publishing Switzerland 2015.
- [16] L. Loeb. Security Intelligence. “Industrial Internet Consortium Develops an IoT Security Framework”. Sep 22nd, 2016. Retrieved October 19th, 2016 from <https://securityintelligence.com/news/industrial-internet-consortium-develops-iot-security-framework/>
- [17] Fkks.com. “The Internet of Things: Best Practices for Developers”. http://fkks.com/news/static_print/the-internet-of-things-best-practices-for-developers. Feb 18, 2015. Retrieved Oct 10th, 2016
- [18] Marco Schwartz. “How to Keep Hackers Out of Your Smart Home”. Feb 1, 2016. Retrieved Oct 10th, 2016 from <https://openhomeautomation.net/how-to-keep-hackers-out-of-your-smart-home/>
- [19] Digi International. (2015). XBee / Xbee-Pro ZigBee RF Modules. User’s Guide. Retrieved October 12, 2015 from www.digi.com/resources/documentation/digidocs/pdfs/90000976.pdf
- [20] Digi International. (2014). Channels, ZigBee. User’s Guide. Retrieved September 5, 2015 from www.digi.com/wiki/developer/index.php/Channels%2C_Zigbee

- [21] P. Dhillon and H. Sadawarti, "A Review Paper on ZigBee (IEEE 802.15.4) Standard", in International Journal of Engineering Research & Technology (IJERT), 2014. ISSN:2278-0181
- [22] Dyn. "Dyn Analysis Summary Of Friday October 21 Attack". Oct 21st, 2016. Retrieved Oct 30th, 2016 from <http://dyn.com/blog/dyn-analysis-summary-of-friday-october-21-attack>
- [23] L.N. Whitehurst, T.R. Aniel, and J.T.McDonald. "Exploring Security in ZigBee Networks", in 9th Cyber and Information Security Research Conference, 2014. ACM 978-1-4503-2812-8/14/4
- [24] N. Borisov. D. Golberg., and D. Wagner. "Intercepting Mobile Communications: The Insecurity of 802.11." Proceedings of the 7th Annual International Conference Mobile Computing and Networking, ACM (2001), pp. 180-189
- [25] J. Durech, M. Franekova. "Security attacks to ZigBee technology and their practical realization", in IEEE 12th International Symposium on Applied Machine Intelligence and Informatics (SAMI 2014)
- [26] S. Farahani. "ZigBee wireless networks and transceivers", Oxford: Newness, Elsevier Inc, 2008. ISBN: 978-0-7506-8393-7
- [27] G. Dini and M. Tiloca. "Considerations on Security in ZigBee Networks", in IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing (2010). pp. 58–65, 2010.
- [28] J. Li and Q. Yang, "I'm a newbie yet I can hack zigbee," in Def Con, Las Vegas, 2015.

- [29] Eduard Kovacs. “Reuse of Cryptographic Keys Exposes Millions of IoT Devices: Study”. Nov. 25th, 2015. Retrieved Oct 20th from <http://www.securityweek.com/reuse-cryptographic-keys-exposes-millions-iot-devices-study>.
- [30] N. Vidgren, K. Haataja, J. L. Patino-Andres, J. J. Ramirez-Sanchis, and P. Toivanen, "Security threats in ZigBee-enabled systems: vulnerability evaluation, practical experiments, countermeasures, and lessons learned," in System Sciences (HICSS), 2013 46th Hawaii International Conference on, 2013, pp. 5132-5138
- [31] CERT Knowledgebase. “Animas OneTouch Ping insulin pump contains multiple vulnerabilities”. Oct 11, 2016. Retrieved Oct 22nd, 2016 from <https://www.kb.cert.org/vuls/id/884840>
- [32] J. Cache, J. Wright, and V. Liu, Hacking Exposed Wireless: Wireless Security Secrets and Solutions, McGraw-Hill, Second Edition, Jul. 2010.
- [33] The Security Ledger. “Botnet of 140,000 Cameras, DVRs Behind Biggest DoS Ever”. Sep. 27, 2016. Retrieved Oct 23rd, 2016 from <https://securityledger.com/2016/09/botnet-of-140000-cameras-dvrs-behind-biggest-dos-ever/>
- [34] X. Cao, D. M. Shila, Yu Cheng, Z. Yang, Y. Zhou, and J. Chen. “Ghost-in-ZigBee: Energy Depletion Attack on ZigBee-Based Wireless Networks”
- [35] Sparkfun. Sparkfun XBee Explorer USB. Retrieved August 13, 2015 from www.sparkfun.com/products/11812

- [36] Digi. XTCU Application. Retrieved August 11, 2015 from <https://www.digi.com/products/xbee-rf-solutions/xctu-software/xctu>
- [37] Arduino. Arduino MEGA 2560. User's Guide. Retrieved August 13, 2015 from www.arduino.cc/en/Main/ArduinoBoardMega2560
- [38] Arduino. XBee Shield. Retrieved August 13, 2015 from www.arduino.cc/en/Main/ArduinoXbeeShield
- [39] Sharp Electronics. (2005). GP2Y0A21YK Optoelectronic Device. Retrieved August 13, 2015 from www.sharpsma.com/webfm_send/1208
- [40] Davy L. AES.Lib. Retrieved August 20, from <https://github.com/DavyLandman/AESLib>
- [41] Node.Js. Node.js v4.5.0 Documentation. Retrieved August 1st, 2015 from www.nodejs.org/dist/latest-v4.x/docs/api/
- [42] Npm. what-is-npm? Retrieved August 1st, 2015 from <https://docs.npmjs.com/getting-started/what-is-npm>
- [43] Socket.io. www.socket.io
- [44] mongoDB. www.mongodb.com
- [45] Angular JS. www.angularjs.org
- [46] Ayeho.com. "When it Comes to IT Security, Incident Response is Key". Jan 12, 2015. Retrieved Oct 10th, 2016 from <http://ayehu.com/when-it-comes-to-it-security-incident-response-is-key/>

Curriculum Vitae

Charbel Azzi

azzic@unlv.nevada.edu

Education

University of Nevada, Las Vegas

Bachelor of Science in Computer Engineering, May 2011

GPA:3.71

University of Nevada, Las Vegas

Master of Science in Computer Science, Dec 2016

GPA:3.44

Thesis: “Vulnerability Analysis and Security Framework for ZigBee Communication in IoT”

Awards

Grand Prize, Harriet & Fred Cox Engineering Design Award, May 2011

Second Place, IEEE Student Paper Contest, May 2011

Second Place, IEEE MicroMouse Competition, Apr 2011

Professional Experience

International Game Technology (IGT), Las Vegas, NV, June 2014 – Present

Software Engineer

- Developing mobile apps and websites using Java, JavaScript, HTML, CSS, NodeJs, Sql, mongoDB, AngularJs, and Polymer.

University of Nevada Las Vegas, Las Vegas, NV, Jan 2013 – June 2014

Graduate Research Assistant

- Developed a website to display the location of sensors on GoogleMaps, visualize the data, and create statistical/performance analysis using PHP, HTML, MySql, JQuery, JavaScript, and GoogleMapsAPI

Intel, Chandler, AZ, Nov 2011- Dec 2012

Software Engineer Intern

- Developed tests for new product features, fixed test suit bugs, and added new UI features for the test suites using C#
- Established a shared UML tool infrastructure to support the engineering team
- Reviewed/updated/verified requirements related to Rapid Storage Technology (RST)