UiO : Department of Informatics
University of Oslo

Building Affordable Intrusion Detection System for Internet of Things

Muhammad Haroon

Master's Thesis Autumn 2018

Building Affordable Intrusion Detection System for Internet of Things

Muhammad Haroon

17th December 2018

Acknowledgement

Foremost, i would like to thank my beloved family for their support and love. I could never do it without them.

I would express my sincere gratitude to my supervisors Josef Noll and Maghsoud Morshedi for their guidance, motivation and patience. I am grateful to have the chance to work with one of the best young talents in UiO, Maghsoud Morshedi.

My special thanks to Hårek Haugerud for his kindness and support.

Finally, i would like to thank my friends and colleagues, specially Madeleine Victoria Kongshavn for her time and valuable suggestions.

Abstract

IoT is a pervasive technology that has revolutionized industry with value added solutions. Internet of Things (IoT) is a collection of small smart devices, which can communicate through communication networks. Due to a high profit potential, it has gained an enormous attention of technology industry. IoT vendors scramble for their profit share in the market, leaving security of these devices almost to negligence. It exposes IoTs to identity theft, privacy breaches and internet downtime. Protecting IoTs has become one of the most researched topics in cyber security paradigm. Due to contemporary nature of IoT devices, it is comparatively difficult and complex to design and implement security software considering constraints attached to IoT. Specifically, an intrusion detection system (IDS) is a crucial tool in IoT security. Several commercial hardware and software products have been claiming to solve these challenges by providing IDS solutions but these solutions are costly and proprietary, which itself is an issue. This thesis presented an affordable intrusion detection system using low price hardware Raspberry Pi (RPi) and open-source IDS Snort. Several tests were conducted to check capability of Raspberry Pi of hosting Snort as an IDS as well as to analyze Snort performance. The results showed that Rpi stood firm against several attack scenarios and Snort gave satisfactory performance. The thesis provided a cost effective solution for IoT security to how to build an affordable intrusion detection system using low price hardware and open-source intrusion detection systems.

Contents

List of Figures viii

List of Tables x

1 Introduction 3
- 1.1 Motivation . . . 4
- 1.2 Problem Statement . . . 5
- 1.3 Thesis Outline . . . 6

2 Background 9
- 2.1 Internet of Things . . . 9
  - 2.1.1 IoT Ecosystem and Trend . . . 10
  - 2.1.2 Areas of IoT Application . . . 11
  - 2.1.3 General Architecture of IoT . . . 13
  - 2.1.4 IoT Protocols . . . 14
  - 2.1.5 Challenges To IoT . . . 15
    - 2.1.5.1 IoT Security Challenges . . . 16
- 2.2 Intrusion Detection System . . . 19
  - 2.2.1 IDS Placement Strategies . . . 20
  - 2.2.2 IDS Detection Methods . . . 21
- 2.3 Open Source IDS . . . 22
  - 2.3.1 Snort . . . 22
    - 2.3.1.1 Snort Architecture . . . 22
    - 2.3.1.2 Configuration Modes . . . 24
    - 2.3.1.3 Snort Rules . . . 25
  - 2.3.2 Bro . . . 26
  - 2.3.3 Kismet . . . 26
  - 2.3.4 Suricata . . . 26
- 2.4 Commercial Intrusion Detection Systems . . . 27
  - 2.4.1 McAfee NSP . . . 27
  - 2.4.2 Trend Micro TippingPoint . . . 27
  - 2.4.3 Hillstone NIPS . . . 27
  - 2.4.4 Huawei NIP . . . 27
- 2.5 Raspberry Pi . . . 28

2.5.1 Core Components . . . 28
2.5.2 Operating System . . . 29
2.5.3 Applications of RPi . . . 29
2.6 ELK Stack . . . 30
2.7 Related Work . . . 31

3 Methodology **34**
3.1 IDS Design Methodology . . . 34
3.2 System Design . . . 35
3.2.1 System Components . . . 35
3.2.1.1 RPi Snort Gateway . . . 36
3.2.1.2 Sensors . . . 36
3.2.1.3 Router . . . 36
3.2.1.4 IDS Test Machine . . . 36
3.2.1.5 Remote Monitoring Server . . . 37
3.3 Snort Algorithms . . . 37
3.3.1 Aho-Corasick State Machine . . . 37
3.3.2 Snort Implementation of AC . . . 40
3.3.3 Detection and Processing of Rules . . . 42
3.4 The Approach . . . 44
3.4.1 Conditions . . . 44
3.4.2 Attack scenarios . . . 44
3.4.3 Monitoring . . . 45

4 Implementation **46**
4.1 Compiling Snort on RPi . . . 46
4.2 Customizing Snort . . . 46
4.3 Snort Instances . . . 47
4.3.1 Snort Instance Eth0 . . . 48
4.3.1.1 Network . . . 48
4.3.1.2 Snort Eth0 Configuration File . . . 48
4.3.1.3 Eth0 Rules File . . . 48
4.3.2 Snort Instance Wlan0 . . . 49
4.3.2.1 Network . . . 49
4.3.2.2 Wlan0 Rules . . . 50
4.3.3 Logging . . . 50
4.4 Configuring Beats on RPi . . . 50
4.5 Configuring ELk server . . . 51
4.6 Testing . . . 51
4.6.1 Test 1 . . . 52
4.6.2 Test 2 . . . 53
4.6.3 Test 3 . . . 54
4.6.4 Test 4 . . . 54

5 Results and Analysis **56**
5.1 Test 1 . . . 56

5.2 Test 2 . . . 58
5.3 Test 3 . . . 60
5.4 Test 4 . . . 61
5.5 Monitoring Server . . . 63
5.5.1 Filebeat . . . 64
5.5.2 Metricbeat . . . 66

6 Discussion **69**
6.1 Project Evaluation . . . 69
6.2 The Approach . . . 69
6.3 Implementation and challenges . . . 70
6.4 Testing and Analysis . . . 71
6.5 Future Work . . . 73

7 Conclusion **74**

Bibliography **75**

Appendices **81**

A Install and Configure Snort **82**

B Hostapd and ISC server configurations **84**

C Configuring Beats on RPi **87**

D Setup ELK on Openstack **89**

E Snort Instances and Gnuplot **93**

F Snort Eth0 configurations **98**

G Snort Eth0 Sample Rules **99**

List of Figures

1.1 *IoT Market Growth Forecast showing IoT market trend in different IoT sectors.* Sources: GrowthEnabler [68] 4

2.1 *major IoT companies and their sector of interest.* Source:[5] . . . 11
2.2 *Global share of IoT projects around the world. It also provides top to lowest IoT segments* Source: IoT Analytics [13] 12
2.3 *General IoT communication architecture, showing most common IoT sensors and devices.* 13
2.4 *A comprehension of major IoT challenges in present and near future.* 16
2.5 *IoT layers in the context of IoT component placement in IoT architecture* Source:[22] 17
2.6 *The most common attacks against IoT Networks.* 19
2.7 *General IDS placement approach, the figure shows a distributed approach IDS placement where IDS sensors placed in distributed locations.* 21
2.8 *An abstracted view of Snort operation process, form packet commencing to output.* 23
2.9 Simple Snort Rule 25
2.10 *Raspberry Pi Model B Block Diagram showing it's main components.* 28
2.11 *Components of ELK stack in their position of operation.* Source:[75] 31

3.1 *Adopted system design with it's components and connection points.* 35
3.2 *Aho-Corasick state machine, showing data input and output steps.* Source[66] 38
3.3 *Form Beats on RPi to ELk server on Openstack.* 45

4.1 *Shows how the Snort is configured on RPi with two Snort instances and Beats* 47
4.2 *Figure created from system design figure, it shows attack vectors on each interface* 52

5.1 Test 1 - *Memory and CPU consumption during test 1 at 100MBit/sec data rate.* . . . 57
5.2 Test 2 - *Memory and CPU consumption during test 2 at 300MBit/sec data rate. CPU consumption is increased but consistent as compared to test 1.* . . . 59
5.3 Test 3 - *Memory and CPU consumption during BlackBurse attack, a very high CPU rate can be noticed with slight decrease in memory consumption.* . . . 61
5.4 Test 4 - *Memory and CPU consumption during dual attack, a very high CPU rate can be seen because both of the Snort instances are operational in detecting attacks.* . . . 63
5.5 Syn-food attack on eth0 - *Snort detects the attack and generates alert, which is sent by Filebeat and processed by ELk index patterns.* 64
5.6 ssh-bruteforce attack on wlan0 - *It shows the generated alert by Snort after detection of the attack.* . . . 65
5.7 blackNurse attack on eth0 - *Snort successfully detects the attack and generates alerts. Figure shows the alert message on the ELK server.* . . . 65
5.8 XMAS scan detected on eth0 - *Figure shows the detection of nmap XMAS scan, which normally belongs to first phases of a hacking attempt.* . . . 65
5.9 GeoIP MAP - *Showing the country location of attackers based on their IP addresses. Red color countries are the top countries where attacks where generated from.* . . . 66
5.10 Memory and system load - *The line graph generated at ELK server using Metricbeat data, showing memory and system load fluctuations during a longer time span.* . . . 67
5.11 Incoming and Dropped packets - *The number of packets received and dropped by Ethernet interface. Based on Data shipped by Metricbeat.* . . . 67

List of Tables

2.1 *Shows an overview of IoT related entities and their respective parameter values.* 14
2.2 *Comparison of RPi alternatives in terms of price and features.* . 30
2.3 *A summary of the major research IoT area with concerned threats and detection methods.* 33

3.1 *Specification details of RPi device used as an IDS.* 36
3.2 *Specification details of IDS testing machine i.e. attacking machine* 37

4.1 *Characteristics of Test 1, syn-flood at 100MBit/sec* 52
4.2 *Test 2 characteristics and it's values for a higher data rate of 300MBit/sec* 53
4.3 *Test 3 characteristics while it used BlackNurse attack at relatively low data rate* 54
4.4 *Test 4 characteristics. It shows a dual attack on both interfaces with different data rates.* 55

5.1 Test 1 - *Snort I/O Statistics, showing Snort analysis and packet drop rate. Percentage values are rounded.* 57
5.2 Test 2 - *Snort I/O Statistics, showing Snort analysis and packet drop rate. Analysis rate has dropped while outstanding rate has increased.* 58
5.3 Test 3 - *Snort I/O Statistics, showing Snort analysis and packet drop rate during BlackNurse attack. A higher packet commence rate by Snort can be noticed.* 60
5.4 Test 4 - *Snort I/O Statistics for eth0, showing Snort statistics during dual interface attack. A very low analysis rate can be seen, with a very high outstanding packet rate.* 62
5.5 Test 4 - *Snort I/O Statistics for wlan0, showing Snort statistics during dual interface attack. A very higher packet commence rate by Snort can be noticed with no packet drop.* 62

Listings

3.1 AC implementation in Snort 41
4.1 Static ip configuration 48
4.2 snort_wlan0.conf 49

Chapter 1

Introduction

Since the very beginning of Internet back in the midst of 1960s at MIT, internet and technology in general, has evolved at a very fast pace, almost exponentially in terms of size, power and functionality.

This tremendous growth has not only changed the world which we now see and interact with, but in fact, it has changed the prospective of human evolution. With all the benefits and good signs, there have been several short comings, which unfortunately could not evolve at the same pace as internet itself, for instance, security of all these inter-connected devices, cyber security to be more precise.

Research in area of cyber security has held a noticeable speed. Due to unforeseen nature of cyber threats, research in this area is always on passive mode, awaiting next cyber-attack to unleash. with advancements in technology, specifically nano technology, computers and other technology related devices has shrunk to minimal size with almost an exponential increase in processing power and communication capability. these multi function smaller devices are capable of data gathering, data processing and communicating remotely over the internet, these devices are generally referred as Internet of Things (IoT) . Direct integration of these devices into the internet made it very difficult to follow traditional cyber security rules.

Unlike traditional computers and internet components, these devices or things, have a direct interaction with people and have a relatively higher degree of impact on quality of life. These devices are referred as Internet of Things (IoTs), term claimed to be first coined by K.Ashton [1] back in 1999. It is the IoT, which holds the triumph to close the gap between day to day activities and internet. It has empowered the internet to such extent that it is very hard to think about a day without internet.

The trend of IoT is growing at a tremendous speed. From baby monitors to

smart homes, autonomous cars to ICS sensors, health services to military operations, IoT will be a vital organ of internet in upcoming era. Imagine, controlling entire home from a cell phone, for instance to control the temperature or monitor the door locks and home security or using smart wearable which monitor health and provide information about blood pressure, EKG, temperature, and oxygen levels from finger or forehead. There are numerous other IoT applications which no wonder, looks like a scene of a Sci-fi movie. IoT has everything to get excited for, but it is also a fact that these devices can be turned into an ultimate nightmare due to insufficient security measures.

1.1 Motivation

According to Forbes [2], The global IoT market is expected to grow from $157B in 2016 to $457B by 2020, attaining a marvelous Compound Annual Growth Rate (CAGR) of 28.5%. Discrete Manufacturing, Transportation and Logistics, and Utilities will lead all industries in IoT spending by 2020, averaging $40B each.

Industrial manufacturing is predicted to increase from $472B in 2014 to $890B in global IoT spending. Health-care and life sciences are projected to increase from $520B in 2014 to $1.335T in 2020, attaining a 17% CAGR. While According to GrowthEnabler & MarketsandMarkets analysis[68], the global IoT market share will be dominated by three sub-sectors; Smart Cities (26%), Industrial IoT (24%) and Connected Health (20%). Followed by Smart Homes (14%), Connected Cars (7%), Smart Utilities (4%) and Wearables (3%).

Similarly, advisory firm Bain predicts the most competitive areas of IoT will be in the enterprise and industrial segments. It is been estimated that consumer applications will generate $150B by 2020, with B2B applications being worth more than $300B. Globally, enthusiasm for the Internet of Things has fueled more than $80B in merger and acquisition (M&A) investments by major vendors and more than $30B in venture capital, according to Bain's estimates. Figure 1.1 shows the year to year growth rate and global IoT market share.

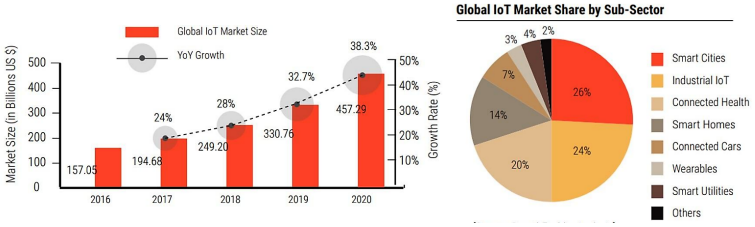

Figure 1.1: *IoT Market Growth Forecast showing IoT market trend in different IoT sectors.* Sources: GrowthEnabler [68]

The statistics and IoT market share predictions are no doubt astonishing but unfortunately the so-called boom of IoT industry is pure profit oriented, overlooking the criticality and security aspects of such devices. Nonetheless, several security companies are being herded by the profit expectancy into the business of IoT.

Researchers in security of these device are trying hard to catch the pace, in fact, it is becoming one of the most researched fields in IT industry. Which is a good sign, but there is too much to be done and several issues are to be addressed. These issues have been the central point of concern for almost all the major IoT stack holders, independent research institutes, government regulations and standardization organizations [3].

To take part in the effort of implementation and deployment of IDS in IoT networks, this thesis is intended to build and test an intrusion detection system for IoT devices, using easy to access, relatively cheaper hardware Raspberry Pi and a light weight open source Intrusion detection tool Snort.

1.2 Problem Statement

There are several commercial IDS (Intrusion Detection System) solutions available for IoTs but these solutions are either too costly or not not designed specifically for IoT environment. Most of these devices are designed for traditional systems, providing only a partial support for IoT devices. It is very important to keep in view the resource constraints of IoT devices when designing an IDS, e.g. IoT devices, have low power and processing capability, are generally deployed in sparse-to-congested and critical areas. Also, commercial IDS solutions are propriety and have cross-platform dependency which makes them restricted to be used in wider range of IoT networks.

This thesis intends to develop an intrusion detection system which is affordable, portable, less power consuming and open-source. The IDS device must be deployable in most of the IoT environments with minimal configuration requirements. To achieve this objective, a 35$ device Raspberry Pi (RPi) with opensource IDS Snort will be used in development of an affordable IDS solution for IoT devices. Moreover, this thesis will evaluate the performance of RPi in terms of resource consumption. The thesis will try to answer following main questions:

a) *How can we build an affordable and portable IDS solution using RPi and Snort?*

b) *Does RPi have enough resources to support Snort as an IDS, in terms of resource consumption?*

Except the core objectives, the thesis should also provide a remote monitoring capability, which will give an insight of Snort alerts as well as performance metrics of RPi. ELK (Elasticsearch, Logstash, Kibana) stack is used as monitoring server on Openstack cloud.

1.3 Thesis Outline

The Thesis is organized as following:

Chapter1 - Introduction

This chapter provides general overview of cyber security, evolution of IoTs and its significance in Tech world. specifically, security aspects of IoTs, its future research directions and what is this thesis aimed for?.

Chapter 2 - Background

This chapter goes through related and involved technologies in detail. A summary of related research work is also presented.

Chapter 3 - Methodology

In this chapter, Research methodology is presented, along with a description of experiment design and result evaluation strategy.

Chapter 4 - Implementation

Chapter 4 provides details of how to practically implemented and test proposed IDS solution.

Chapter 5 - Results and Analysis

This chapter analyzes the experiment results and provides the justification of the results.

Chapter 6 - Discussion

A critical discussion on analysis and results, comprising the limitations and strengths of the proposed work as well as future work.

Chapter 7 - Conclusion

Concluding the thesis by connecting problem statement and how the thesis have addressed the issues outlined.

Chapter 2

Background

This chapter will provide a technical overview of the tools, technologies and all the main components which will be involved or relevant to this project. With an emphasis on Internet of Things IoTs, Intrusion detection systems, as well as a review of related research material will be discussed and elaborated.

2.1 Internet of Things

Following the tradition of innovation in recent years, birth of Internet of Things (IoT) can be tracked back to Massachusetts institute of Technology MIT. Auto-ID center, founded in 1999, was assigned a task to work on a sensing technology called RFID (radio frequency identification). Auto-ID was responsible to design the first ever architecture of IoT. IoT generally, refers to a very large collection of small smart devices connected to internet.

The phenomenal growth of smart-phones and tablet PCs sling shot the number of connected devices to an approx. of 12.5 billion in 2010, for a population of 6.8 billion, increasing the ration from 0.08 to 1.84 devices per person. Cisco IBSG predicts that there will be 50 billion devices connected to the internet by 2020[4]. It is worth a mention that estimates do not take into consideration the fast-paced evolution and advancements in device technology. Secondly, entire population is taken into calculation, in fact, a large portion of population is still not yet connected to the internet by a true sense of internet connection I.e. when it starts influencing their quality of life.

2.1.1 IoT Ecosystem and Trend

After the development of wireless sensor network (WSN) technology, several standardization technologies are formed, based on interest in market trend and technology selection. Technically, WSN can subdivided into IP based and non-IP-based solutions. Most of non-IP based solutions belong to well-known alliances like, Zigbee [47] and WAVE2M [48] for office manufacturing, automation, wirelessHart [49] and PROFIBUS [50] for industrial control systems (ICS). The heterogeneous communication system of these alliances can produce several issues.

For the rest of IP-based devices, Internet Engineering Task Force (IETF) leads the standardization convoy to propose and develop standard protocols by keeping in view the limitations and characteristics of concerned devices. For instance, IPV6 over Low Power Wireless Personal area Network (6LoWPAN) [6], routing protocol for low power and lousy networks (RPL) [7] and Constrained Application Protocol (CoAP) [8].

Most of these protocols are aimed towards coping with protocols overhead, energy and memory consumption, and computational limitations of IoT devices [9].Several other institutions are working on standardization of IoT, for instance, IP Smart Object Alliance (IPSO) promotes IPv6 embedded devices for machine -to-machine (M2M) applications, PROFINET works on Ethernet standards for industrial automation.

Regardless of the fact that most of IoTs are heterogeneous at lower protocol layers but they do share common behavior at application layer. This similar behavior leads to standardize application layer of IoT which is independent of underlaying communication networks. This responsibility is taken by European Telecommunication Standards Institute (ETSI) Technical Committee (TC M2M) [10]. ETSI will have to specify service requirements, use cases, functional architecture and communication interfaces. To provide the basis of one horizontal IoT platform, ETSI will bring common service layer to globally accepted common agreed grounds.

Although several non-government entities are playing a major role in standardization but there are still some areas where governments have to assist them in this regard. For instance, fragmented solutions and interoperability across vertical applications. More specifically, IoT related solutions are not well accepted by the customers, severe privacy and security concerns are the major factors of customers hesitation. For these reasons, governments have to take their role to rectify the entire ecosystem of IoTs. For instance, US regulation authority have provided a detailed action plan to standardize the development of smart grids [11]. Similarly, in Europe, eCall service for vehicle fitment is to be standardize. While in China, IoT standardization and development has secured its place in China's 12th five-year-plan [12]. Figure 2.1 provides an overview of major IoT market players.

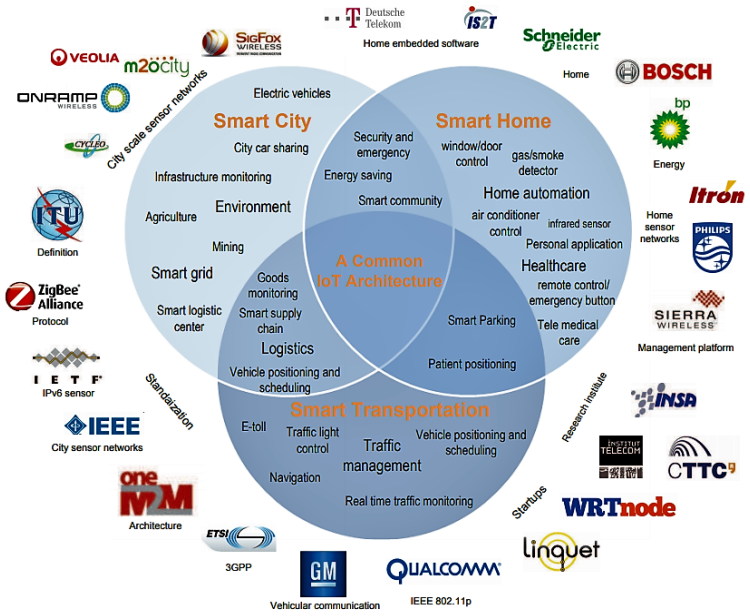

Figure 2.1: *major IoT companies and their sector of interest.* Source:[5]

2.1.2 Areas of IoT Application

A recent report issued by iot-analytics [13] provides a real-life analysis of IoT statistics, based on the data collected from 1600 IoT projects worldwide. The report asserts that 367 projects are aimed for smart city, 265 for industrial settings and connected building IoT projects secures a 3rd place, in a list of top 10 ongoing projects. Americans are ahead in the race by a share of 45% in total IoT projects, EU is at 35% and while rest of Asia contributes only for 16%.

It is also interesting to note that most of smart city projects are located in Europe, almost 45%, while Americans are mainly focusing on connected health 55% and connected cars at 54% in north America. Asia is keener towards smart Agriculture products i.e. 31%. Figure 2.2 depicts the results, showing share of IoT projects around the globe in different IoT sectors.

Some of the main IoT segments are explained as follows:

a) **Smart City -** By 2025, with more than 60% of the world population expected to live in urban cities. By 2023, there will be 30 mega cities globally, with 55% in developing countries, such as China, India, Russia and Latin America [14]. Smart city has become the number one IoT application area in 2018 as per report by IoT analytics. The immense growth in this sector is due to the hundreds of projects initiated by private sector and governments

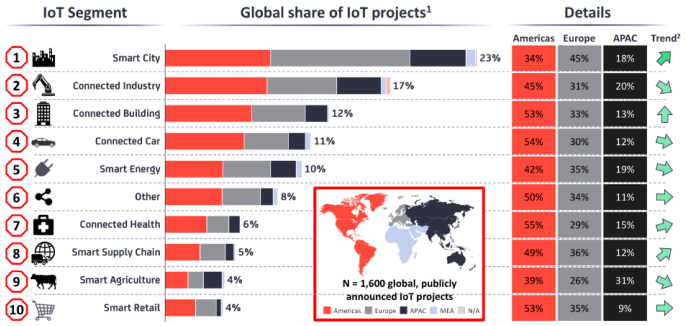

Figure 2.2: *Global share of IoT projects around the world. It also provides top to lowest IoT segments* Source: IoT Analytics [13]

around the world.

For instance, Singapore and Barcelona are the prominent examples in this area. In the segment of smart cities, biggest area of focus is smart traffic, comprising projects on monitoring and control of traffic, bike sharing, parking systems and smart buss lane while related minor projects include utilities, smart lighting, public safety and environmental monitoring. At the current time i.e. 2018, Europe accounts for the highest smart city initiatives.

b) **Connected Industry -** This area of IoT covers a vast range of heterogeneous applications and devices. Equipment monitoring, and non-factory environments have been a prime focus. Non-factory applications include asset monitoring, and management of remote machinery such as drills, heavy cranes, oil ridges and mining components etc.
Cisco's connected mining operation for Rio Tinto in Australia is a vivid example. Other mentionable connected industry applications include production floor monitoring, remote PLC control and automated quality control system. For instance, Varroc, is using digital factory solutions from Altizon to connect industrial machines in the manufacturing environment [15]. However, America plays the leading role in connected industry scenario.

c) **Connected Building and Connected Car -** Connected Building and Cars – connected building has gained the largest increase in number of projects since 2016 i.e. 7%. A majority of projects are aiming for facility automation to reduce energy cost.
For instance, Marriott hotel in China is utilizing building automation solutions, which are supposed to reduce energy cost up to 15%. Rest of the projects are related to building security and to heating, ventilation and air conditioning HVAC. Connected cars has been also a top favorite for IoT projects because it provides a wide-open ground for experimentation. According to IoT analytics, vehicle diagnostics and fleet management are the prominent points of application i.e. 77% and 57% respectively. A recent example in this

context is TracknStop, an Ireland based software company has developed a vehicle diagnostic solution which enables real-time tracking, sensor reading, monitoring and remote control for vehicles.

2.1.3 General Architecture of IoT

There is no universally agreed architecture of IoT networks, due to the fact that it is highly use case dependent. Every organization who wish to deploy an IoT network, has the freedom to design and implement according to their use case and capacity.

Figure 2.1 presents the heterogeneous nature IoT technologies, where each deployment vertical seems unique and independent, but in fact, they do share a lot of similarities at multiple layers of their operation. Similarly, many of network elements and several architectural components share certain common properties. This helps generalize the architecture of IoT, despite a varied nature of IoT entities. It can also be considered as a basic blue print on to which, a variety of architectural verticals can be designed and deployed.

Figure 2.3 presents a generalized architecture of IoT, which shows some of the main components involved in communication and data processing.

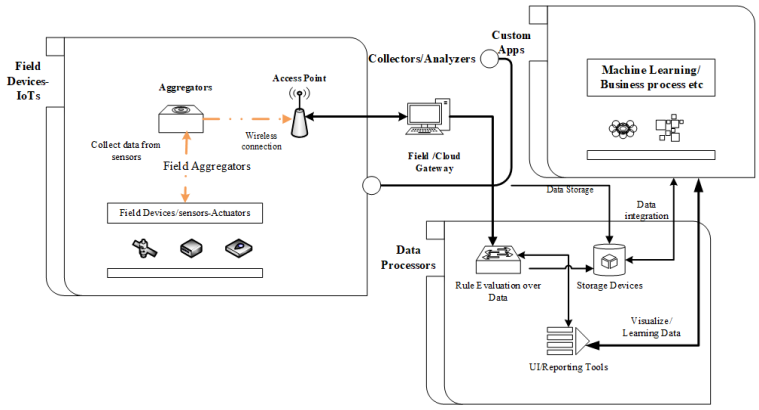

Figure 2.3: *General IoT communication architecture, showing most common IoT sensors and devices.*

Some of the mutual attributes and challenges are summarized as follows:

1. Minimal to no human direct interaction of field IoT devices make it essential to provide a reliable remote communication and management system.
2. In a large network, a huge amount of data is to be processed and analyzed, which makes it important to provide data storing, mining and sharing devices.

3. In case of smart cities or industrial applications, a complex network is to be expected, enforcing the need to resolve addressing and congestion issues.
4. Large IoT networks are generally heterogeneous in nature, which means it demands appropriate naming and addressing mechanisms, specifically for QoS requirements.
5. With appropriate management system, it is also vital to deploy an efficient remote monitoring system, which must provide a visual status of all network devices.

2.1.4 IoT Protocols

As it has been discussed in earlier sections that IoTs tend to inherent a heterogeneous nature, both in functionality and operability. IoT mostly communicate through wireless medium, which is more feasible due to their physical attributes and environments IoTs operate in. In this regard, designing a standalone protocol which can fulfill all the IoT specific requirements is a difficult task. At the moment, no such standalone protocol exists.

There is a plethora of protocols and related technologies underlie the IoT technology. some of the main data protocols will be briefed in following section. Table 2.1 presents the layered approach towards the IoT technologies, which categorizes a variety of opensource and proprietary solutions.

Entities	Identity
Infrastructure	6LowPAN, IPv4/IPv6, RPL, NanoIP, DTLS, TSMP etc.
Identification	EPC, uCode, IPv6, URIs
Transport/Communication	Wifi, LoRa, 802.15.4, Sigfox, Z-wave, Zigbee, WirelessHArt, NB-IoT
Discovery	Physical web, mDNS, DNS-SD, UPnP, HyperCat
Data Protocols	MQTT, CoAP, AMQP, Websocket, SMCP, XMPP, DDS, REST, LLAP etc.
Device Management	TR-069, OMA-DM
Semantic	JSON-LD, Web Thing Model
Multi-layer Frameworks	Alljoyn, IoTivity,Weave, Homekit

Table 2.1: *Shows an overview of IoT related entities and their respective parameter values.*

a) **MQTT -** (Message Queening Telemetry Transport) is a standard protocol provided by Organization for the Advancement of Structured Information Standards (OASIS) [16]. It is a lightweight publish/subscribe messaging

transport application level protocol. It is designed for IoTs and machine-to-machine communication. MQTT uses TCP/IP for communication and it can provide a reliable point to point connection stream. For networking, it uses Ethernet, WiFi and cellular networks such as, 3G, 4G etc.

A common MQTT structure consists of three main components, publisher, broker and subscriber. Publisher is usually a small sensor device which sends data to broker. Subscribers are the applications connected to broker and send the collected data from broker to IT infrastructure. Main goal of MQTT is to provide remote monitoring i.e. telemetry, as its name implies.

Amazon now supports MQTT in their Amazon Web Services2, over both IPv4 and IPv6 [17].

b) **CoAP -** (Constrained Application Protocol) is designed by Internet Engineering Task Force IETF to facilitate message transfer between machine-to-machine applications by offering various features such as, built in discovery, multi cast support and asynchronous message exchange.[18].

c) **WebSocket -** as like CoAP, Websocket protocol is also designed by IEFT [19]. It was developed as part of HTML5 initiative by implementing a JavaScript interface, which facilitates full duplex communication between client and server. The WebSocket standard simplifies much of the complexity around bi-directional web communication and connection management.

d) **XMPP -** (Extensible Messaging and Presence Protocol) Extensible Messaging and Presence Protocol (XMPP) is an open XML technology for real-time communication, which powers a wide range of applications including instant messaging, presence, collaboration, multi-part chat, voice and video calls, content syndication and generalized routing to XML data [20]. it is designed to be extensible.

2.1.5 Challenges To IoT

Regardless the fact that the concept of IoT is very new, but it is rather immature. Surprising enough, it has grown its own ecosystem, comprising of ever-growing billions of heterogeneous devices, numerous protocols, almost non-uniform architectures, several parent/manufacturer organizations (mostly start-ups) with prime goal of mere profit, different function classes i.e. application from room temperature sensors to sensors in outer space, all of this diversity spawn challenges in every component of this ecosystem.

It is true, if said, that currently, IoT is in its non-equilibrium state. To prove its true worth and without making things more miserable, challenges to IoTs must be rectified, or at least their effect must be minimized. Luckily, there have been several initiatives taken by public and private organizations to provide solutions and to raise awareness.

Challenges associated with IoT range from social issues to frightening security concerns. For instance, *Ethical issues*, which requires to explain user rights, raising public awareness etc. *Legal Issues*, demanding whether certain laws have to be enforced or not, user consent, legal use of devices etc. *Economic*, which is mostly a concern for IoT producers or the benefactors of IoT in terms of direct profit.

It is directly connected with Legal issues on the long run. IoT business entities must provide reasonable Terms of Service, honest marketing! and a clear business model. Last but not least, the most important, *Technical and Security* Challenges. Figure 2.4 depicts the scene, showing major issues associated with IoTs.

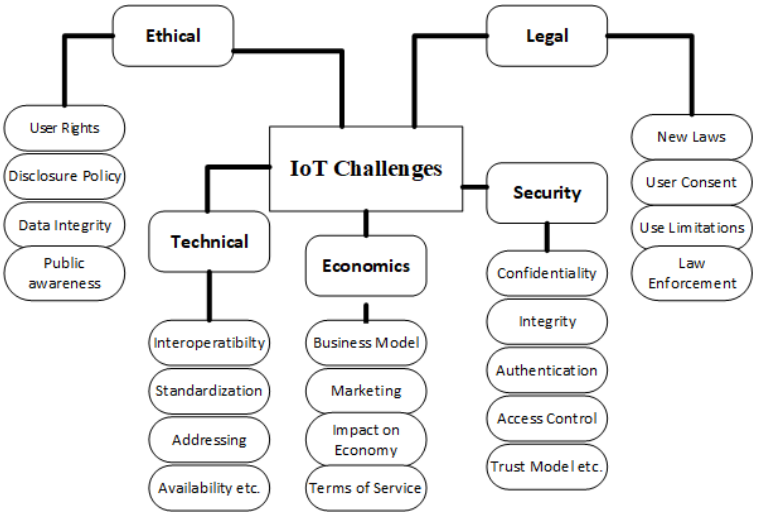

Figure 2.4: *A comprehension of major IoT challenges in present and near future.*

To reap the greatest from IoT, rectification of Technical and Security challenges is vital. Since the thesis is aimed for rectification of one of the security problem, only security related issues will be discussed in next section.

2.1.5.1 IoT Security Challenges

A generic IoT system can be defined by using concept of layers i.e. perception, transportation and application layer. These layers come along with their own type of weakness and security vulnerabilities. In [21], authors have analyzed IoT security issues with possible security solutions. Figure 2.5 depicts the layered security perspective of IoTs, showing security challenges associated with IoT on different layers.

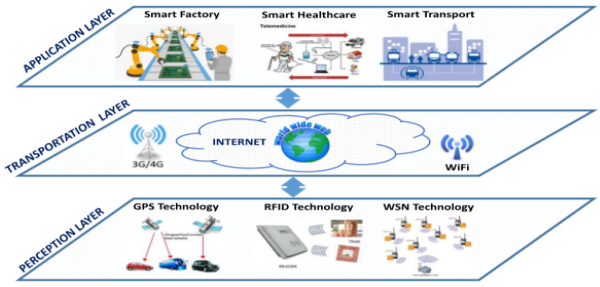

Figure 2.5: *IoT layers in the context of IoT component placement in IoT architecture* Source:[22]

I. Perception Layer
Perception layer holds the physical features and components on IoT infrastructure. It can be logically compared to physical layer of OSI model. It provides features of data collection and processing on different communication technologies, i.e. radio frequency identification (RFID), Wireless Sensor Network (WSN).These devices are generally deployed randomly without security considerations. Due to limited node resources and broadcast nature of these devices several security threats may result. Few of them are described as follows.

   (a) Physical Attacks - Physical attacks, as the name explains itself, are the close proximity attacks. Physical attacks can be, For instance, by changing original node with a malicious node, connecting to ports physically, tempering with circuits and electronics, using Electromagnetic pulse (EMP) to disrupt communication.

   (b) Data Transit Attacks - During the data transmission, integrity and confidentiality of the data can be compromised, for instance, by Man in The Middle (MITM) attack, or by sniffing the broadcasted data transmission.

   (c) Impersonation Attacks - Authentication methods in IoT devices are hard to implement. It is very trickier to detect an imposter node with fake spoofed identity.

   (d) DoS - A DoS attack can be unleashed on this layer by battery draining i.e an attacker sending lots of packets, outage attacks, or when an edge device stops performing its normal operation.

II. Transportation Layer
This layer is responsible to provide data transportation facilities to Perception layer entities. It transmits the gathered information using communication networks (WiFi, Ad-hoc, 4G etc.) to an information processing system. Few of the major threats at this layer are as follows:

(a) Routing Attacks – A node infected with a malware, may modify the actual routing and forwarding tables, enabling the node to forward the data to its peer system controlled by attacker, e.g. to a remote Command and Control Center (C&C).

(b) DoS - this layer can also be compromised by a DoS. For example, jam the transmission of radio signals.

(c) Injecting fraudulent packets – Done via insertion (where malicious packets that seem legit are generated and sent), manipulation (when packets are captured then modified) and replication (where the attacker captures packets between two things to replay them).

III. Application Layer
The application layer provides the services requested by customers. For instance, the application layer can provide temperature and air humidity measurements to corresponding users. The importance of this layer for the IoT is that it has the ability to provide and present IoT services in an easy to use manner, to meet the requirements of a user.

Many different IoT environments i.e. smart city, smart healthcare, and smart factory can be implemented within this level. Moreover, an application support sublayer, to support all sorts of business services and to realize intelligent computation and resources allocation, could be implemented throughout specific middleware and cloud computing platforms. Some of the major attacks related to Application Layer are explained below:

(a) Information Leakage - A vulnerable application software can easily lead to confidential information leakage. For instance, username and passwords, Emails, home addresses, Credit Card info etc. application software are notorious for bugs and vulnerabilities.

(b) Script Injection - Attackers can upload malicious codes in software applications exploiting the known or unknown vulnerabilities. e.g. exploiting zero day vulnerabilities.

(c) DDoS - Distributed Denial of Service, is the most frequent and famous attack on application layer. It targets weak spots in Web applications and uses various techniques to compromise the application availability. One of the famous methods adopted by DDoS is using Botnets. A **botnet** is generally a large collection of compromised IoT devices. These compromised IoT devices work as a sleeping agent, keep waiting for commands from their master or C&C center. A specially written malware is used to create botnets. When needed, the botnets are signaled by their master to send data traffic towards a specific target. DDoS attacks on application layer are evolving quickly, which make these types of attacks very hard to detect and mitigate.

Figure 2.6 shows some of most common conventional attacks on different layers of IoT operation. The Open Web Application Security Project (OWASP)

Layer	Main Threats
Application Level	Data Leakage
	DoS Attacks
	Malicious Code Injection
Transportation Level	Routing Attacks
	DoS Attacks
	Data Transit Attacks
Perception Level	Physical Attacks
	Impersonation
	DoS Attacks
	Routing Attacks (e.g. in WSN, RSN)
	Data Transit Attacks (in WSN or RSN)

Figure 2.6: *The most common attacks against IoT Networks.*

also have a useful list of IoT Attack surface areas which they state should be understood by manufacturers, developers, researchers and companies looking to deploy IoT in their organizations [23].

Other IoT specific attacks are *sink hole attack,* where all traffic is directed from an area through a compromised node, where selective forwarding can follow with the attacker deciding what data to allow through.*Sybil attack,* a single node presents multiple identities to others in the network, so an attacker can be in more than one place at once. *Wormhole attack,* an attacker tunnels messages received in one part of the network over a low latency link and replays them in a different part. These type of attacks were first mentioned in [24].

2.2 Intrusion Detection System

Networks and information system are subject to get infected and hacked. Due to free availability of vulnerability analysis tools, the trend of attacks has risen. Tools such as SubSeven [52], BackOrifce [53], Nmap [54], L0phtCrack [55] can all be used to scan, identify, probe and penetrate information systems.

An intrusion detection system is supposed to detect an intrusion and generate appropriate alert. IDS accomplish this by collecting information from different system and network resources. After collection of information, it analyzes the data for any suspicious behavior or security issue and generates alert. It can be a hardware or a software, providing a variety of options to customize and implement.

An intrusion detection system provides following core functionality:

- Monitoring and analysis of user and system activity
- Auditing of system configurations and vulnerabilities
- Assessing the integrity of critical system and data files
- Statistical analysis of activity patterns based on the matching to known attacks
- Abnormal activity analysis
- Operating system audit

According to SANS [25], There are two main types of an intrusion detection system as follows:

Network Intrusion Detection system (NIDS) performs an analysis for a passing traffic on the entire subnet. Works in a promiscuous mode, and matches the traffic that is passed on the subnets to the library of knows attacks. Once the attack is identified, or abnormal behavior is sensed, the alert can be send to the administrator. Example of the NIDS would be installing it on the subnet where the firewalls are located in order to see if someone is trying to break into the firewall.

Host Intrusion Detection System (HIDS) takes a snap shot of existing system files and matches it to the previous snap shot. If the critical system files were modified or deleted, the alert is sent to the administrator to investigate. The example of the HIDS can be seen on the mission critical machines, that are not expected to change their configuration.

2.2.1 IDS Placement Strategies

Correct placement of IDS into network infrastructure is important. Although it is largely dependent on the environment and needs of the network, most common places to deploy an IDS can be identified. Some of the recommended placement points for IDS are illustrated as:

- Between network and Extranet
- In the DMZ before the firewall to identify the attacks on servers in DMZ
- Between the firewall and network, to identify a threat in case of the firewall penetration.
- In the Remote access environment
- If possible between servers and user community, to identify the attacks from the inside.
- On the intranet, FTP and database environment.

Figure 2.7 depicts the typical NIDS placement on a network, showing multiple possible deployment points for IDS sensors.

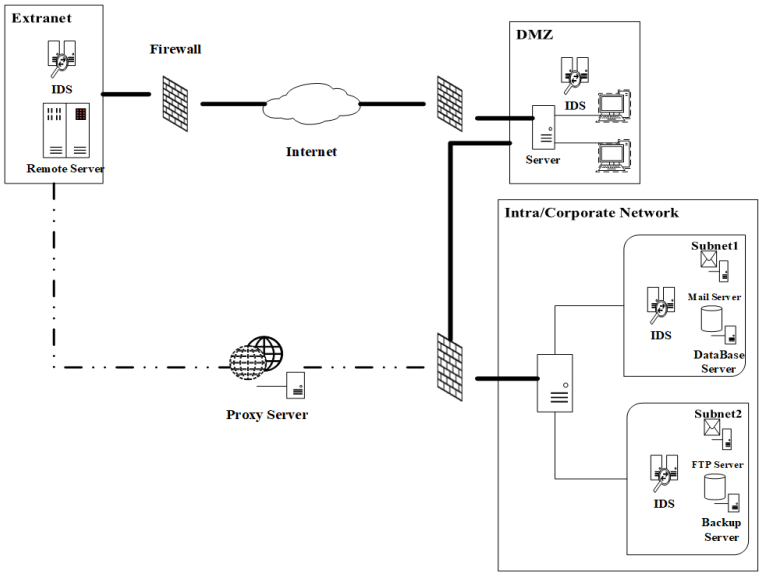

Figure 2.7: *General IDS placement approach, the figure shows a distributed approach IDS placement where IDS sensors placed in distributed locations.*

In terms of IDS placements, there are two typical methods, Distributed and parallel IDS placement. In Distributed technique, IDS sensor is placed in strategic locations around the network, where each IDS node is reporting their monitoring status to a centralized IDS manager. The goal of distributed IDS is to improve the quality of detection process and minimize the risk of intrusions [26].

While in parallel or centralized placement, IDS monitor network traffic on the single ingress/egress point on the network and processes the network traffic in parallel or concurrently. The goal of parallelism in IDS is to reduce processing time when doing pattern matching against packets, which in result increases throughput.

2.2.2 IDS Detection Methods

An IDS has two major categories in terms of detection, signature based and anomaly-based detection methods. In signature-based method, IDS use a data base of patterns or signatures of previously known attacks and then matches the traffic with these patterns, in case of a match, it generates an alert. This method is highly accurate with comparatively least number of false positives, but it is

effective against known attacks only. It will be unable to detect unknown or zero-day attacks.

Anomaly based detection uses behavioral knowledge as its data base for intrusion detection i.e. it needs a collection of statistics and facts of a normal system behavior. Whenever it notices any deviation in normal behavior it raises an alert. This method is hard to implement as it requires machine learning techniques and system training. Since there can be an unknown number of normal behaviors, it is susceptible to a higher degree of false positives.

2.3 Open Source IDS

There are several open source intrusion detection systems available over the web, each with its strengths and weakness. Since snort is part of this thesis project, it will be presented in a good detail. While the other IDS systems will be just briefly introduced.

2.3.1 Snort

Snort is an open source network intrusion detection and prevention system, capable of performing real-time traffic analysis and packet logging on IP networks. It can perform protocol analysis, content searching/matching, and can be used to detect a variety of attacks and probes, such as buffer overflows, stealth port scans, CGI attacks, SMB probes, OS fingerprinting attempts [27]. it was created by Martin Roesch in 1998 and is now maintained by Sourcefire Snort Team.

2.3.1.1 Snort Architecture

Snort functions by taking a series of steps and procedures. A typical snort procedure involves packet sniffing, Packet Decoder/Acquisition, Pre-processor, Detection Engine and Output. Although it depends on the mode of deployment, but a general architecture of snort can be depicted. Figure 2.8 shows snort operation architecture. The figure shows a typical Snort architecture consisting of it's operational steps.

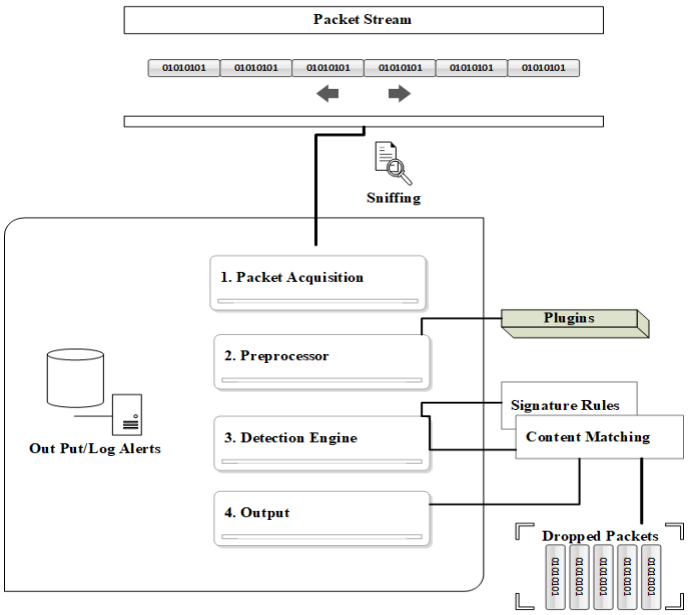

Figure 2.8: *An abstracted view of Snort operation process, form packet commencing to output.*

Packet Acquisition is the first step in intrusion detection, in this phase each packet is captured and identified by its structure. It uses libpcap library to capture packets, as Snort does not inherently posses such capability. After capturing the packets, these packets are then passed on to the next phase called preprocessor.

Preprocessor is a collection of modular plugins and is one of the core component of Snort. There are several preprocessors offered by snort, which are categorized in two types by the mode of their operation. First type is responsible to do complex tasks by adding an extra layer of analysis. This is done in case if signature-based detection cannot be translated into rules to detect intrusion. The second type allows the modular plugins analyze the packets for suspicious behavior and modify them accordingly, so that the detection engine could interpret these packets with accuracy. For instance, malformed or obfuscated URL goes through prepossesses for normalization such that pattern matching could be performed with precision, with minimal risk of sneaking through detection engine.

Detection Engine is the main detection module, it performs signature detection and parsing based on the database of rules it contains. For faster processing these rules are loaded into memory. After arrival of a packet, detection engine inspects the packets by matching its header and payload with loaded rules in memory. Snort uses first match then exit logic, which means when a

rule matches it generates the alert and then moves to next packet inspection. This phase is notorious for memory and processing consumption, which makes it a difficult choice for IoTs. During a heavy traffic flow, this phase leads to a higher packet drop.

Output works below the detection engine and in fact, it is a component of detection engine itself. The Output stage logs intrusion alerts that were triggered by the detection engine. These logs can be saved in different databases that can be used for third party security information and event management systems like Snorby [56], ACID [57], ELK Stack [58] or BASE [59] which provide a visual and understandable output than the logs in raw form.

2.3.1.2 Configuration Modes

Snort can be configured to run into three different mods, namely, sniffer mode, packet logger mode and network intrusion detection mode. These modes are further discussed as follows:

- Sniffer mode is configured to sniff the ongoing traffic, without generating alerts and applying any rules, logically, it can be compared with Tcpdump [60], while ran as sniffer. Advantage of using Snort in sniffing mode over other sniffer tools is that it provides a good summary of captured packets which helps for an easy troubleshooting.
- When Snort runs in packet logger mode, it collects every packet it sees and places it in a directory hierarchy based upon the IP address of one of the hosts in the datagram. After specifying a logging directory Snort can easily be run as logger mode by running *./snort -dev -l ./log or ./snort -l ./log -b* . -b switch tells the snort to log traffic into a single binary file, into Tcpdump binary format. It is very easy to read these binary formatted logs by using any sniffer, such as, Tcpdump or Ethereal. Snort can also read the packets back by using the -r switch, which puts it into playback mode.

  It is possible to manipulate the data in the file in a number of ways through Snort's packet logging and intrusion detection modes, as well as with the BPF interface that's available from the command line. For example, to see the ICMP packets from the log file, simply specify a BPF filter at the command line and Snort will only see the ICMP packets in the file i.e. *./snort -dvr packet.log icmp.*
- Network intrusion detection system mode provides threat detection and analysis, by using a database of rules and filters. to enable Network Intrusion Detection System (NIDS) mode, use *./snort -d -l ./log -c snort.conf* command, this will configure Snort to run in its most basic NIDS form,

logging packets that trigger rules specified in the snort.conf in plain ASCII to disk using a hierarchical directory structure.

2.3.1.3 Snort Rules

Snort uses a simple, lightweight rules description language that is flexible and quite powerful. These rules can be downloaded from snort community database. These databases are updated periodically with disclosure of new threats and vulnerabilities. It is also possible to customize the rules according one's own needs.

Snort rules are divided into two logical sections, the rule header and the rule options. The rule header contains the rule's action, protocol, source and destination IP addresses, netmask and the source/destination ports information. The rule option section holds alert messages and a detailed description, which is used by detection engine to determine that which parts of the packet should be inspected to decide if the rule action should be taken. Figure 2.9 shows a simple snort rule: explaining sequentially from left to right:

```
alert icmp any any -> $HOME_NET any (msg: "ICMP test"; sid:1000001;
rev:1; classtype:icmp-event;)
```

Figure 2.9: Simple Snort Rule

Rule Header

- alert - Rule action, Snort will generate an alert when the set condition is met.
- any - Source IP, Snort will look at all sources.
- any - Source port, Snort will look at all ports.
- -> - Direction, From source to destination.
- $HOME_NET - Destination IP, using the HOME_NET value from the snort.conf file.
- any - Destination port, Snort will look at all ports on the protected network.

Instead alert option, other options are drop and log, which provide Snort inline functionality and logging respectively.

Rule Options

- msg:"ICMP test" - Snort will include this message with the alert.
- sid:1000001 - Snort rule ID, all numbers less than 1,000,000 are reserved. for custom rules, any arbitrary number above that limit can be used.
- rev:1 - Revision number, This option allows for easier rule maintenance and is used for version control.
- classtype:icmp-event – Categorizes the rule as an "icmp-event", one of the predefined Snort categories. This option helps with rule organization.

These rules are highly flexible in terms of customization. Following the syntax, a Snort rule can be written in a variety of ways with multiple option to increase detection accuracy. for instance, Pearl compatible regular expressions PCREs can also be used.

2.3.2 Bro

Bro is an open-source network traffic analyzer. It is primarily a security monitor that inspects all traffic on a link in depth for signs of suspicious activity. Bro supports a wide range of traffic analysis tasks even outside of the security domain, including performance measurements and helping with trouble-shooting [28].

2.3.3 Kismet

Kismet is a network detector, packet sniffer and an intrusion detection system 802.11 wireless LANs[81]. It can be used with almost any wireless card which support monitoring mode. It can detect both wireless access points and wireless clients and their mutual association. It has the ability to log sniffed packets and save them in pcap formats for later analysis. It can detect non-configured networks, probe requests, and level of wireless encryption used on a given access point.

It consists of three main parts, a drone, which can be used to collect packets and pass them to server, a server for interpretation of received packets, and the client which communicates with the server and displays the information server collects.

2.3.4 Suricata

Suricata is an open source intrusion detection system (IDS) and intrusion prevention system (IPS). It was developed by the Open Information Security Foundation (OISF) and was first released in January 2010. It is a rule based IP/ID

system that utilizes externally developed rule sets to monitor network traffic and provide alerts to the system administrator when suspicious events occur. it features unified output functionality and Plug&Play library options to accept calls from other applications[29].

2.4 Commercial Intrusion Detection Systems

There are several Commercial ID/IP systems available. some major products are as follows:

2.4.1 McAfee NSP

The McAfee Network Security Platform (NSP) [61], is a network threat and intrusion prevention solution that protects systems. It can support up to 32 million connections on a single appliance uses intelligence to find and block advanced targeted attacks on the network. a typical NSP setup can cost around $10,000.

2.4.2 Trend Micro TippingPoint

TippingPoint [62] identifies and blocks malicious traffic, prevents malware infections, ensures network availability and resiliency, and enhances network performance. The solution offers network traffic inspection throughput up to 120 Gbps. Starting price of TippingPoint T Series is $6000, which can go as high as $72,000.

2.4.3 Hillstone NIPS

The Hillstone Network-based IPS (NIPS) [63] appliance offers intrusion prevention, anti-virus, application control, advanced threat detection, abnormal behavior detection, a cloud sandbox and a cloud-based security management and analytic platform. It can identify more than 3,000 applications, including mobile and cloud. It's package price starts from $18,000.

2.4.4 Huawei NIP

Huawei Network Intelligent Protection (NIP) [64] provides virtual patches, web application protection, client application protection, anti-malware, antivirus, anti-DDoS, application sensing, control on IPv4 and IPv6 networks. It's price starts from $2,000.

Commercial intrusion detection systems can be deployed in large corporations. Because of very high pricing, these commercial solutions are not feasible for medium to small organizations. An open source solution with a proper configuration can also satisfy security needs of an organization. In [30], authors have provided a good comparison of open source and commercial IDSs.

2.5 Raspberry Pi

The Raspberry Pi [65], also referred as RPi, is a single board minicomputer developed at United Kingdom by Raspberry Pi Foundation for the purpose of promoting computer science education in developing countries. It was primarily aimed to be a low cost device but powerful enough to support several experimental scenarios, ranging from small school projects to robotics. Figure 2.10 shows the block diagram of Raspberry Pi model B.

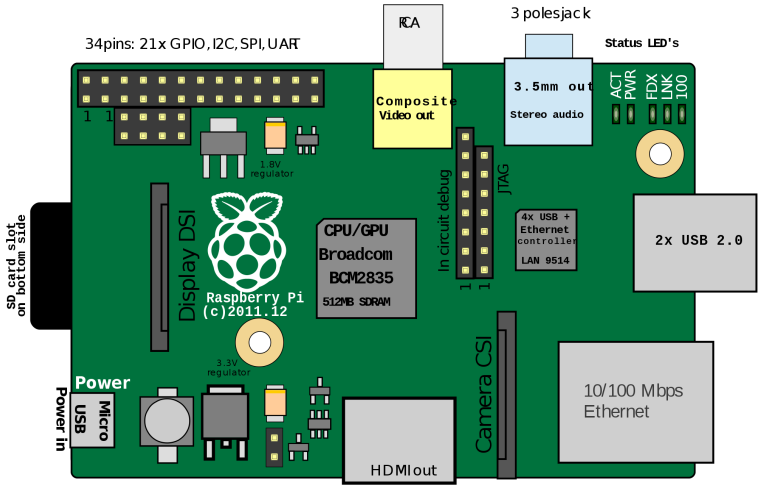

Figure 2.10: *Raspberry Pi Model B Block Diagram showing it's main components.*

2.5.1 Core Components

Some of the main components of Raspberry Pi are introduced below:

Processor

The first generation RPis were equipped with Broadcom BCM2835 system of chip (SoC), which included a 700 MHz ARM1176JZF-S processor, VideoCore

IV graphics processing unit (GPU) and RAM. It has a level 1 (L1) cache of 16 KB and a level 2 (L2) cache of 128 KB. The level 2 cache is used primarily by the GPU. The SoC is stacked underneath the RAM chip.

Memory

Original B boards were allocated 256 MB of split-able memory, which means 128 MB was allocated to GPU, while the rest was reserved for CPU. In further releases of model B board, a dynamic memory allocation method was adopted, which could dynamically split memory to GPU and CPU. Raspberry Pi 3 B+ comes with 1 GB of RAM memory.

Networking Components

The Raspberry Pi 3 and Pi Zero W (wireless) have 2.4 GHz WiFi 802.11n (150 Mbit/s) and Bluetooth 4.1 (24 Mbit/s) based on the Broadcom BCM43438 FullMAC chip. Raspberry Pi 3 B+ comes with dual-band IEEE 802.11b/g/n/ac WiFi and Gigabit Ethernet supporting up to 300 Mbit/sec.

2.5.2 Operating System

By default, Raspberry Pi is native to Raspbian, a Debian-based Linux distribution. It is available for download from RPi Foundation's website [65] for free. It also offers several other choices of operating system for RPi, e.g. Ubuntu MATE, RISC OS and Windows 10 IoT Core. Except these, there are numerous operating systems supported by RPi which includes both non-Linux based and Linux-based operating systems.

2.5.3 Applications of RPi

Due to its small size, very low price and power full customization features, RPi has been the most favorite technology for researchers, science teachers, automation industries and independent science enthusiast. A RPi can be turned into almost any anything if its components support it. For instance, prototyping of electrical equipment was a tedious and costly job, now it can be done by mere $35 of RPi, by utilizing its GPIO and peripherals.

It's applications range from media server, arcade Machine, personal security system for network devices, a mini computer, automation, robotics and countless other. It had made the life easier for all those technology related people who are looking for cheap and customize-able solutions.

There are several alternative single board computers available in the market. Depending on the features and price, these computers can be a good alternative to RPi, for instance, some offer a higher CPU and memory and are equipped with higher bandwidth Ethernet ports. Table 2.2 shows some of the alternatives that can be chosen as an alternative to RPi.

Name	CPU	Memory	Ethernet	Price
Orange Pi Plus2[76]	1.6GHz	2GB	Gigabit	49$
Asus Tinker Board[77]	1.8GHz	2GB	Gigabit	60$
Banana Pi M3[78]	1.8GHz	2GB	Gigabit	72$
Firefly RK3288 Plus[79]	1.8GHz	4GB	Gigabit	79$
armStoneA9[80]	1.2GHz	4GB	Gigabit	330$

Table 2.2: *Comparison of RPi alternatives in terms of price and features.*

2.6 ELK Stack

ELK, also known as Elastic stack is acronym for three open source projects, Elasticsearch, Logstash, and Kibana[75]. Elasticsearch is search and analytics engine. Logstash is a data processing pipeline which can receive data from several different sources simultaneously, parse and transform it and finally hand it over to a stash like Elasticsearch. Kibana is the visualization tool, which visualizes data with interactive charts and graphs in Elasticsearch. Later in never releases, ELK stack also integrated lightweight single-purpose data shippers called Beats.

Elastic stack has provided an end-to-end delivery system which can provide real time analysis of almost any type of structured and unstructured data source. Together, these different open-source products are used for centralized logging in IT environments. Except centralized logging, Elastic stack has numerous other applications including business intelligence, web analytics and security and compliance.

In a common setup, Beats collect the data and ship it to Logstash which collects and parses logs with provided patterns, Elasticsearch indexes and stores the information. Kibana enables presentation of the data with live and interactive graphs and charts. Figure 2.11 presents the layers of Elastic stack with Kibana on top of Elasticsearch while Beasts and Logstash are on the base, connected directly to data feeds.

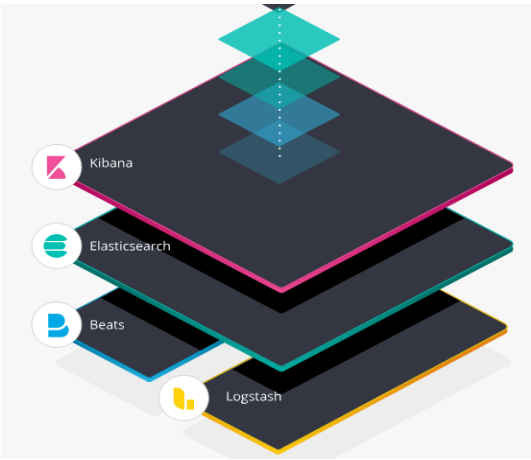

Figure 2.11: *Components of ELK stack in their position of operation.* Source:[75]

2.7 Related Work

In recent years, IoT has been the hot topic for research, specifically security of IoT has been researched and presented in numerous articles. These articles have tried to cover various aspects of IDSs in IoT networks, for instance, [31,32,33] have proposed IDS strategies for mobile and ad-hoc networks (MANETs), [34,35,36] have provided IDS proposals for Wireless Sensor Networks, and [37] have presented IDS placement methods in cyber-physical systems (CPS).

The articles referenced above are aimed mainly for the design of IDSs for IoT related technologies but not specifically IoT networks itself. In [38], Zarpelao et al. provided a thorough study of IDS techniques which is more specific to IoT networks and IoT devices. Articles realted to IoT security can be categorized as, detection method, placement strategy and security threat. In the following section, some of the most significant and closely related research work to this thesis project is presented:

E. Cho, et al. [39] have proposed a centralized IDS approach, where an IDS sensor is placed in border router such that data traffic pass through physical and network domain. These packets are then inspected for possible botnet traffic. They adopted anomaly-based approach and was based on the assumptions that botnets cause unexpected change in pattern flow of IoT sensors. For normal behavior profiling they used an average of three normal behavior metrics. When any node violated the average behavior profile, an alert is raised.

R. Chen et al. [40] have presented signature-based intrusion detection ap-

proach which is based on artificial immune system mechanisms. They deployed sensors with information of attack signatures, referred as immune cells. These cells are said to be able to classify datagrams as malicious or normal. However, their paper does not provide any information about the placement strategy neither it concerns that how this method is feasible for a resource constraint IoT network. Collecting data for artificial immune cells can be a computational and resource overhead.

Raza et al. [41] proposed an IDS solution for IoT named SVELTE. The solution is aimed for specific types of attack against IoT networks i.e. sinkhole and selective forwarding attacks. There method deploys a hybrid placement strategy where border router and network nodes are involved in attack detection. Boarder router is responsible for detecting intrusion by analyzing RPL network data, while network nodes are programmed to send information to the border router, reporting any malicious traffic and sending RPL network data. It also adopts hybrid approach in detection methods i.e. anomaly-based and signature-based to balance computational cost and storage cost respectively.

Lee et al. [42] published about a lightweight IDS system for IoTs. The paper adopts a distributed placement strategy which monitors energy consumption of nodes for detecting intrusion, i.e. if energy consumption is above a normal threshold an alert is raised. This is said to be useful against Dos attacks or if a node is a part of a botnet. Moreover, a node is classified as malicious if it consumes above the normal threshold energy and is removed from the routing table. Authors used anomaly-based approach to analyze node behavior by energy consumption. Since their method focuses only on a single node parameter i.e. energy consumption, less computational overhead is expected.

Cervantes et al. [43] proposed an IDS solution for IoT named INTI (Intrusion detection of Sinkhole attacks in 6LoWPAN for Internet of Things). In their method, nodes are placed in a hierarchal structure leading to a distributed placement strategy. Each node in the hierarchy is assigned a role, where a superior node is monitored for its traffic patterns. It combines the specification-based method by utilizing trust and reputation of nodes and anomaly-based method to monitor exchange of packets between nodes. If, in case, a node detects a sinkhole attack, it broadcasts a message to alert the other nodes.

A. Le et al. [44], presented a lightweight IDS solution for IoTs. They adopted a hybrid strategy by dividing the network into small clusters. Each cluster is assigned a cluster head which is responsible for communicating with all cluster members. Cluster head is installed with an IDS sensor and it monitors all of its cluster members, while member report only to their respective cluster head. Border router is also installed with an IDS sensor and is responsible for more resource consuming instances. The authors adopted specification-based approach for detection of routing attacks.

Knowledge-driven Adaptable Lightweight Intrusion Detection System (Kalis), is an IDS approach presented by **Midi et al.** [45]. Kalis can be deployed on

border router or on an external device as a standalone tool. It uses a centralized placement approach. While in detection approach, it is hybrid in nature, such that it is self-adapting knowledge-driven IoT intrusion detection system which can work on different communication protocols.

Kalis is said to be autonomous in collecting information about network features and its involved components. Then using this information, Kalis dynamically configures the most effective set of detection techniques. It can be extended for new protocols standards while the collected information can be shared for collaborative detection. Compared to traditional IDS, authors claim that their system is capable of detecting DoS, routing and conventional attacks.

A. Sforzin et al. [46] presented an IDS solution for IoTs using Raspberry Pi and Snort. Their system is aimed for scalability, robustness, ease of use and versatility. It is claimed to be an effectively portable on demand IDS that notifies the users or the administrators of the network. Whenever it detects an ongoing attack or suspicious network activities.

They adopted a centralized approach where RPi acts as a border router while installed with snort sensor. All alerts can then be directed to remote server running a Security and event Management software (SIEM). To check resource consumption of Snort, they used different rule sets of snorts, so that only the most relevant rules to IoT could be loaded. Authors conclude that their system is not well suited for higher data rates and larger networks.

Table 2.3 summarizes some of the major efforts in IoT-IDS applications.

Research	Placement approach	Detection method	Threat
E. Cho, et al. [39]	Centralized	Anomaly-based	Botnet
R. Chen et al. [40]	Hybrid	Spicification-based	Routing attacks
Raza et al. [41]	Hybrid	Hybrid	Routing attacks
Lee et al. [42]	Distributed	Anomaly-based	DoS
Cervantes et al. [43]	Distributed	Hybrid	Routing attacks
A. Le et al. [44]	Hybrid	Specification-based	Routing attacks
Midi et al. [45]	Centralized	Hybrid	Conventional attacks
A. Sforzin et al. [46]	Centralized	Signature-based	Conventional attacks

Table 2.3: *A summary of the major research IoT area with concerned threats and detection methods.*

Chapter 3

Methodology

This chapter outlines approaches and steps towards building an intrusion detection system using affordable and portable hardware i.e. Raspberry Pi and open source intrusion detection system (IDS) Snort. In first phase, this chapter will explain intrusion detection design methodologies, system design and it's components. While in next phase, adopted Snort algorithms will be presented.

3.1 IDS Design Methodology

It is very important to be vigilant while selecting IDS strategy. Specially, when dealing with a resource constrained environment, for example, in an IoT scenario. Typical IDS methodology consist of intrusion detection method, IDS placement approach, target attacks and validation method. This thesis adopted following IDS methods:

i) **Detection Method -** A typical intrusion detection method can be signature-based, anomaly-based, specification-based or a hybrid of these methods. Since Snort uses it's rule database to inspect and match pre-defined signatures of traffic, this thesis adopted signature based detection method.

ii) **IDS Placement Approach -** The thesis adopted centralized placement approach, Snort was placed in Raspberry Pi gateway. All the connected IoT devices or sensors were connected to internet through the gateway's access point only.

iii) **Targeted Attacks -** Conventional attacks hold a much larger bucket of attacks vectors, ranging from DoS, bots, scanners, malwares etc. these are the attacks which are more frequent and devastating.

This thesis was mainly concerning with conventional attacks, with a focus on DoS, detecting botnet traffic, alerting against scanners and recognizing ssh dictionary attacks. A custom snort rule file was developed for each of the interface i.e. eth0 and wlan0.

3.2 System Design

The system was designed to keep in view the complications of IoT environment. Since IoT devices are wireless and have a broadcast signaling nature, it is relatively complicated to design a stand alone security mechanism for IoT devices. For test purposes, several assumptions were to be made, for instance, assuming that only certain kind of IoT devices with same protocols are used e.g. sensors with Message Queuing Telemetry Transport (MQTT).

Figure 3.1 shows the system design architecture. It gives an abstract idea of how the system components were connected to each other. It also shows position of intrusion detection device and how the attack machine and remote ELK (Elasticsearch, Logstash, Kibana) server were connected.

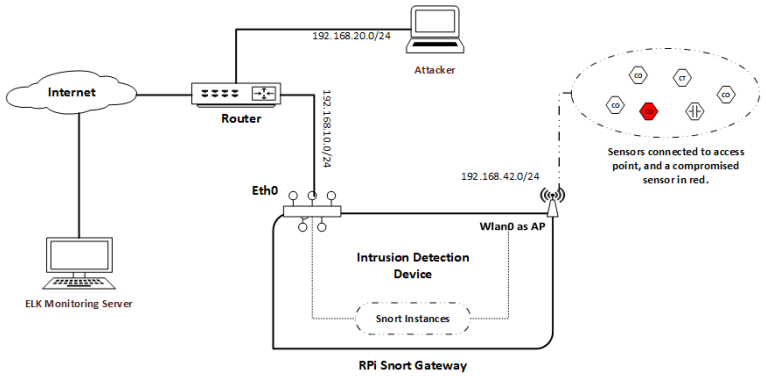

Figure 3.1: *Adopted system design with it's components and connection points.*

3.2.1 System Components

As it can be seen from previous chapters, this thesis used empirical approach (using physical systems and not simulations) to deploy and test the system components. Some of the core components of the setup are explained in the following section.

3.2.1.1 RPi Snort Gateway

Raspberry Pi 3 model B+ was used as a core gateway equipped with Snort sensor. Integrated Wi-Fi of RPi was used as hotspot to connect remote sensors. For operating system, Raspbian stretch 9.4 was installed. Table 3.1 shows details about Raspberry Pi, used as a gateway Snort sensor.

Spec	**Raspberry Pi 3 B+**
CPU type/speed	ARM Cortex-A53 1.4GHz
RAM	1GB SRAM
Ethernet speed	300Mbps
Operating System	Raspbian stretch 9.4
Snort Version	2.9.12 IDS mode

Table 3.1: *Specification details of RPi device used as an IDS.*

Snort version 2.9.12 was used as stand alone intrusion detection system. It was further customized to meet the resource constrained environment of RPi gateway. It's customization details are provided in chapter four. Moreover, Snort was configured to operate in IDS mode.

3.2.1.2 Sensors

An another RPi was used to attach sensors with. This was a Raspberry Pi model B+ with Raspbian lite operating system. It was isolated and deployed at fairly distant location. Moreover, it was connected to RPi Snort sensor gateway's access point by using integrated Wi-Fi module.

3.2.1.3 Router

It is very important to isolate Snort Gateway and attack machines from each other and to make the attacks realistic as well as controllable. For this purpose, all the deployed systems were placed in different networks through static network configuration. It was accomplished by using MikroTik hEX router series. This small and cheap router provides five 100/1000 Ethernet ports which can provide up to a Gigabit throughput.

3.2.1.4 IDS Test Machine

Kali Linux [74] was chosen as operating system for IDS testing due to its built-in penetrating testing packages. It is one of the most sophisticated Linux distro

which provides hundreds of test options for digital forensics and testing capability of any security setup. It was installed on a HP core i5 laptop. Table 3.2 shows some of the important specification of IDS testing machine.

Spec	value
CPU type/speed	Intel Core i5-3210M CPU@2.50GHz x 4
RAM	8GB
Ethernet speed	1000Mb/s
Operating System	Kali Rolling 2018.4 64 bit

Table 3.2: *Specification details of IDS testing machine i.e. attacking machine*

3.2.1.5 Remote Monitoring Server

Monitoring server was setup on Openstack cloud. It is an Ubuntu LTS machine loaded with Logstash, Elasticsearch, and Kibana (ELK stack). For secure remote access to Kibana dashboard, Nginx reverse proxy was placed. It can be accessed from anywhere after provision of correct username and password.

3.3 Snort Algorithms

Snort is a single threaded pattern matching intrusion detection system which relies merely on Aho-Corasick [66] multi-pattern matching algorithm. Aho-Corasick is a backbone of Snort search and detection mechanism. Several variants of this algorithm are available to be used in Snort, where each has it's strengths and weaknesses as well as each one has a significant impact on performance and resource utilization. In next section, Aho-Corasick (AC) is introduced briefly.

3.3.1 Aho-Corasick State Machine

The original AC algorithm was first presented by Alfred V. Aho and Margaret J.Corasick (Aho et al.) in 1975 [66]. It was designed to locate all occurrences of any of a finite number of keywords in a string of text. It is based on the construction of a finite state pattern matching machine from the keywords and then use this pattern matching machine to process the text string in a single pass.

For example, lets K = $\{y_1, y_2, ...y_k\}$ be a finite set of strings called *keywords* while lets x be an arbitrary string called *text string*, Then purpose of state machine is to locate and identify all sub-strings of x which are keywords in

K. overlapping is permitted for substring x. The pattern matching machine consists of several states where each state is represented by a number. The machine operates by processing each text string x by reading every symbol in x and generating output. The state machine consists of three major functions: a goto function g, a failure function f and an output function *output*.

Figure 3.2 represents the state machine and the three functions. It takes set of example keywords {he,she his, hers}.It has been explained by authors Aho et al. as follows:

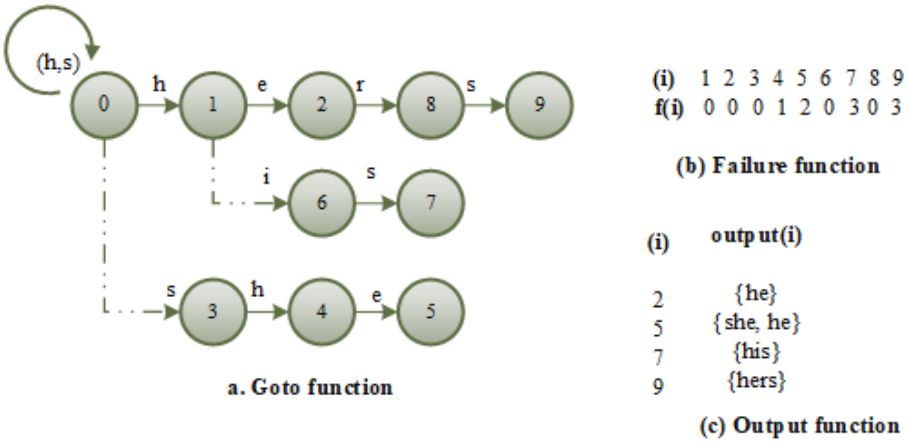

Figure 3.2: *Aho-Corasick state machine, showing data input and output steps.* Source[66]

In goto function, state 0 (zero) is the start state, going thorough till state 9. Goto function g is responsible for mapping symbols in a pair or otherwise move to failure function. Moreover, in goto function the edge labeled as h from state 0 to state 1 means that $g(0,h) = 1$, while absence of arrow means a fail state.

The failure function f is responsible to map states to each other. Whenever goto function generates a failure, function f is consulted. As it's name indicated, function *Output* is responsible for generating output in case of a success symbol match. Figure 3.2 can be translated into pseudocode as illustrated by the Aho

et al., pseudocode is given in below section.

Algorithm 1: Aho-Corasick pattern matching machine

```
1  Input. Any text string consisting of input symbols, e.g. x = a1a2.....an.
     where ai is an input symbol. And a pattern matching machine M with
     goto function g, failure function f and an output function output.
2  Output. Location of keyword x.
3  begin
4    state ←0
5    for i ←1 until n do
6      begin
7        while g(state, ai) = fail do state ←f(state)
8        state ←g(state, ai)
9        if output(state) ≠ empty   then
10         begin
11           print i
12           print output(state)
13         end
14     end
15 end
```

The failure transitions can be removed by Algorithm 1 by introducing functions of deterministic finite automate (DFA) in-place of *goto* and *failure* functions.

When an input character stream is provided, a DFA requires only one transition of the state machine to locate the correct next state. It consists of a finite set of states S, a next move function δ, where for each state s, and input symbol a, δ(s, a) is a state in S.

In Algorithm 1, the DFA next move function δ can be used instead *goto* function. For instance, this can be done by replacing first two statements in *for-loop* by a single statement $state \leftarrow \delta(\text{state}, a_i)$. Aho et al. have presented the DFA version of AC algorithm as given in Algorithm 2. They concluded that amount of memory required to store δ in Algorithm 2 is higher than the corresponding *goto* function. Whereas using DFA in Algorithm 1 can reduce number of state transitions to as low as 50% of total state transition in original implementation.

Snort implements optimized version of Aho-Corasick. There are several variants available to choose from. Each of the variant has it's own performance and memory trade-offs. The optimized version uses state transition table, which is a list of pointers to the state vectors. Where each state vector represents a valid state transition. Moreover, each state vector includes a header and a Boolean flag, indicating storage format of the row and occurrence of pattern matching respectively.

There has been an extensive research on improvement of Aho-Corasick for Snort, to improve match speed as well as to minimize memory usage.

Algorithm 2: DFA version of AC algorithm

```
16 begin
17   queue ←empty
18   for each symbol a do
19     begin
20       δ(0, a) ←g(0, a)
21       if g(0, a) ≠ 0 then queue ←queue ∪ {g(0, a)}
22     end
23   while queue ≠ empty do
24     begin
25       let r be the next state in queue
26       queue ←queue - {r}
27       for each symbol a do
28         if g(r, a) = s ≠ fail do
29           begin
30             queue ←queue ∪ {s}
31             δ(r, a) ←s
32           end
33         else δ(r, a) ← δ(f(r), a)
34     end
35 end
```

3.3.2 Snort Implementation of AC

Snort team leader Marc Norton in [67], tested and analyzed the optimized version of AC algorithm in Snort with it's different variants, where he found significant performance and memory usage differences. As compared to the original version, optimized version handles the state table differently i.e. it breaks the state table into transition tables, where each contains a pattern matching list and a separate failure pointer list, which improved both memory usage as well as match speed. Optimized AC algorithm can be presented in a simple code as given in the Listing 3.1.

The *NextState* in Listing 3.1, is an array of pointers to the row vectors of the state table, while *ps* points to the current state's transition vector. The first two words in array *ps* represent storage format and boolean match flag. The parameter *T* is the input text being searched, while *xlatcase* is case converter which converts the text to uppercase on byte at a time. It is used in case independent scenarios. When a match is found, the pattern is processed. To verify that a pattern has been successfully matched, *if(ps[1])* is used as a check statement. In case of no match, control sequences moves to the next state, while in case of a match, all the patterns in current state are processed in *for* loop.

```
for (state = 0; T < Tend; T++)

   {
     ps = NextState[state];
     sindex = xlatcase[T[0]];
     if(ps[1])
     {
       for (mlist = MatchList[state];
            mlist != NULL;
            mlist = mlist -> next)
        {
              /*process the pattern match*/
        }
     }
   state = ps[2u + sindex];
   }
```

Listing 3.1: AC implementation in Snort

Snort Version 2.9.12 provided various choices of choosing pattern matcher algorithms. The search methods or searching algorithms are categorized in two areas, queued match search methods and non-queued search method. Moreover, both of these methods provide their own search method variants. Some of the search method options are described in following section.

Queued match search method

In this method all the matched pattern are queued in cache until pattern matcher is done scanning the payload. Afterwards these matched cases are evaluated. This method provides following options to choose from:

- **ac and ac-q -** This method is also known as Aho-Corasick full. It provides the best performance available with highest memory usage. This method is not recommended for resource constrained environments, as it is very resource demanding.
- **ac-bnfa and ac-bnfa-q -** This is one of the best methods available. This method is based on binary NFA (non-deterministic finite automata.) It provides a high performance with low memory usage. Although this method does not provide as good as first method but it works good for most of the environments.
- **lowmem and lowmem -** This method is based on *low memory keyword trie*. It provides moderate matching performance with minimal memory usage. This method is recommended for most resource constrained environments where Snort sniffing interface is not exposed to a higher attack

vector i.e. towards internet but is exposed to only limited attack vectors e.g. an access point exposed to a limited connectivity area.

- **ac-split -** This is the default pattern matcher algorithm implemented in Snort version 2.9.12. It provides a high matching performance with low memory usage. Compared to ac-bnafa-q, it consumes more memory. Snort recommends this method for most of the machines. This method is known also as split-any-any.

Non-queue search method

In this method, as it's name indicates, all the patterns are processed individually without queuing in cache. *nq* option specifies that matches should not be queued and evaluated as they are found. It provides following pattern matcher options:

- **ac-nq -** This method is same as ac-q but without queuing. It provides best performance with high memory usage.
- **ac-bnfa-nq -** In previous versions of Snort, this was the default search method. It provides high performance with low memory usage without queuing.
- **lowmem-nq -** It is same as lowmem-q without queuing capability.

Default installation of Snort is not recommended, specially while dealing with resource constraint scenarios. This thesis has implemented Snort in an IoT environment on a relatively low spec device i.e. Raspberry Pi. Snort is customized with several configurations to match the hardware components capability without sacrificing performance and memory utilization. Chapter four will provide the Snort configuration details.

3.3.3 Detection and Processing of Rules

Snort was functioning in IDS mode. Detection and rule processing phase is highly resource consuming because of pattern matcher algorithms. For example, in case of syn flooding attack, Snort first sniffs the packet, sends it to detection engine, which processes the rule with pattern matcher algorithm which inspects packet payload and header information and matches it with both the rule option and rule header i.e. it looks for Source and destination IPs and ports in rule header. Next, it looks for rule options, for instance, if SYN flag (S) is set, and the interface received more than 20 packets with flag *S* within thirty second time period, match is successful. Snort detection engine then passes the packet to output module which takes further configured action. Algorithm 3 shows how a SYN flooding rule is processed by Snort.

Algorithm 3: SYN flooding rule processed by Snort

```
1  Input. Acquired traffic packets during sniffing.
2  Output. Configured action, alert, log, drop.
3  Start
4    load config-file
5  Initialize
6    {
7      output plugins;
8      pre-processors;
9      plugins;
10     rule files;
11     }
12   begin
13     {
14     packet acquisition;
15     packet processing;
16       }
17   Case: (rule matching; instance: detection engine)
18       for s in packet do:          # s is any string in packet
19         {
20           if
21             {
22               source ip == $EXTERNAL_NET;
23               port == $PORT;
24               flags == S;
25               packet count == 20;
26               time == 30 seconds;
27               }
28           do:
29             {
30             generate alert;
31               }
32           else:
33             {
34               ignore;
35               commence next packet;
36               }
37           end
38           }
39       end
```

3.4 The Approach

In this thesis project, system components were chosen and configured carefully to avoid unnecessary IoT environmental factors which can effect system test results. Several conditions were placed to avoid factors which can negatively influence test results.

3.4.1 Conditions

The conditions regarding components connectivity and attack scenarios are given as follows:

- All the sensors were connected to the RPi gateway access point only and there was no other connectivity point for the sensors. Also, sensors were configured with static IP addresses. For internet connectivity, RPi gateway was the only route available to all the sensors.
- All the senors were of same type and were configured over another Raspberry Pi. MQTT was the only telemetry protocol which had been implemented for gateway-sensor communication. SSH was also enabled between gateway and the sensors (via another RPi).
- RPi gateway was not exposed to the internet. Instead, it was connected to MikroTik router. This step was taken to avoid unnecessary traffic from internet, which can influence test results negatively.
- It is literally impossible to test all possible attack vectors. Only few of the major and more common attacks were chosen for system testing, e.g. DoS, SSH dictionary attacks and port scanners.

3.4.2 Attack scenarios

There were two possible attack points on RPi gateway, i.e. interface eth0 and wlan0. The eth0 interface provided internet connectivity through the router while wlan0 acted as an access point and provided connectivity to the sensors in WiFi range. Both of the interfaces had different attack vectors because each interface was exposed to different attack spectrum. For instance, eth0 was prone to a larger attack vector than wlan0 as it was exposed towards internet. On contrary, wlan0 had a smaller exposure area for direct attacks i.e. WiFi signal range.

Keeping this phenomenon in view, two instances of Snort were running, one on each interface. Each Snort instance was provided with different rule files according to their attack vectors, i.e. eth0 needs more rules than wlan0. Also, both of the instances were using different pattern matching algorithms.

To evaluate the Snort detection rate and RPi's resource usage, different levels of DoS were initiated by variation in data rate in MBit/s. Moreover, in further test, multiple attacks on both of the interfaces were initiated to evaluate the capability of Snort and RPi.

3.4.3 Monitoring

In case of successful intrusion detection, each interface must log the alerts into ELK (Elasticsearch, Logstash, Kibana) remote server. In case of local logging, both of the interfaces sent logs to their respective separate log files. ELK was used as monitoring server on Openstack cloud which can receive, analyze, and visualize the logs in real time. Except Snort logs, RPi performance logs were also shipped to ELk, to analyze the performance of Raspberry Pi during the attack.

Filebeat and Metricbeat are the data shippers which were configured on RPi to collect performance data and logs and ship them to ELK server on Openstack which was configured to receive these logs through Logstash, which process them and the forwards these logs to Elasticsearch for storage. For visualization Kibana was used, which receives data from Elasticsearch and visualizes it. Kibana can be accessed through Nginx proxy server after providing username and password. Figure 3.3 shows the steps of data transfer from RPi to ELK server .

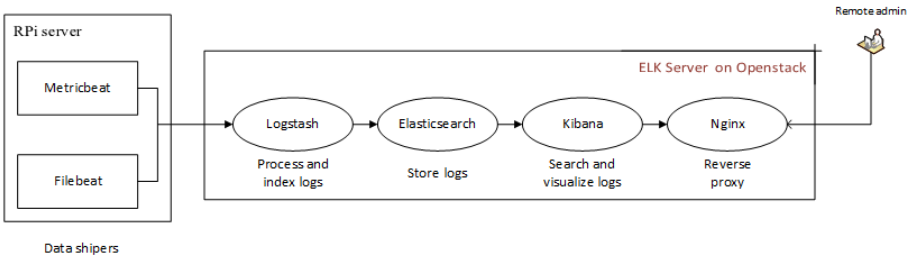

Figure 3.3: *Form Beats on RPi to ELk server on Openstack.*

Data related to memory usage and CPU consumption was collected and sent to monitoring server. It can provide vital information about the RPi resource utilization and generated Snort alerts. Monitoring system facilitates the performance evaluation of Snort running on RPi.

Chapter 4

Implementation

This chapter guides through the process of implementing system design. It provides details about Snort and Raspberry Pi configurations and system testing.

4.1 Compiling Snort on RPi

Installation of Snort in RPi device is relatively trickier because of Arm Linux architecture dependencies. Due to these dependencies, several compilation errors can occur, to avoid such errors, it is better to compile Snort from source. It is recommended not to install Snort as *root*, instead use *sudo* on a non-root user base. Appendix A shows how Snort was configured on Raspberry Pi.

It is a good idea to disable all the default rule paths. Following *sed* command can be used to disable rule paths.

```
sudo sed -i "s/include \$RULE\_PATH/#include \$RULE\_PATH/" /etc/snort/snort.conf
```

4.2 Customizing Snort

If Snort is installed with default configurations and with default rule sets, it can easily overwhelm even a powerful server. For example, there are several open source portals which provide ready to use Snort rules. e.g. EmergingThreat[69], Talos[70] and Snort community-rules[71]. Collectively, these portals provide 15 to 20 thousand rules, which are updated with new rules almost everyday. When these rules are loaded into memory and Snort starts to process them with patter matcher algorithm (assume ac-split Snort default) one by one for a potential match, it can apparently claim a large memory chunk, letting the system paralyzed for processing other crucial system tasks.

This thesis configured Snort for RPi with selected rules and ac-bnfa-nq algorithm. Figure 4.1 gives an abstract idea of how Snort was configured and what areas of Snort were tweaked and tuned for RPi. The following section provides further details of Figure 4.1.

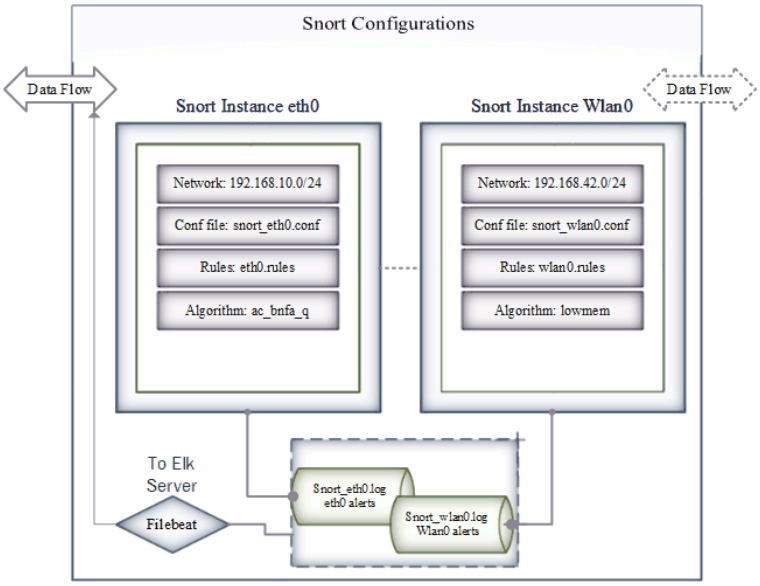

Figure 4.1: *Shows how the Snort is configured on RPi with two Snort instances and Beats*

4.3 Snort Instances

Two instances of Snort were running, one was sniffing over eth0 interface while another on wlan0 interface. It may sound unreasonable to run two instances of Snort on a low spec device such as RPi but the major issue is that Snort is single threaded which means in IDS mode it can only sniff a single instance at a given time, for sniffing on multiple interface, multiple Snort instances must be used, one for each interface.

Both of the instances were loaded with separate configuration files and rule sets. Snort instances can be run either on separate terminals or as a daemon. To run Snort as daemon, for instance on eth0, type the command as follows:

```
sudo snort -d -c /etc/snort/snort.conf -l /var/log/snort -D -i eth0
```

The *-D* switch runs Snort as a background process while -l instructs Snort to start logging to the specified directory. Snort background instance can be killed

by *ps -eaf | grep snort* and then *pkill -9 PID* (Snort PID). Following section further explains how Snort eth0 instance was configured.

4.3.1 Snort Instance Eth0

Interface eth0 was responsible for provision of internet connectivity to whole network. A snort instance was configured to run on interface eth0 with specific custom configurations and rules. following sections provide details about eth0 configurations.

4.3.1.1 Network

Using Mikrotek router, interface ethernet2 was configured for Raspberry Pi network 192.168.10.0/24 with subnet-mask 255.255.255.0 and default gateway as 192.168.10.1. RPi's eth0 was provided static ip address 192.168.10.20. It is important to not that raspbian-stretch RPi OS does not support static ip configurations in directory /etc/network/interface, but picks static ip configurations from */etc/dhcpcd.conf* file. Static configuration for eth0 can be seen in the following code.

```
interface eth0
static ip_address=192.168.10.20
static routers=192.168.10.1
static domain_name_servers= 8.8.8.8 192.168.10.1
```

Listing 4.1: Static ip configuration

No filtering or firewall was enabled on router to minimize test effecting data.

4.3.1.2 Snort Eth0 Configuration File

A separate Snort configuration file for interface eth0 was created and loaded with eth0 specific configurations. Appendix F shows the configurations of snort_eth0 file for interface eth0. Enabling perfmonitor is optional, but it is a good idea to enable it, because it gives real time Snort performance metrics. These metrics can either be saved to an output file or to console. In Appendix F, pfmonitor is set to output data to console when Snort exits.

4.3.1.3 Eth0 Rules File

Since eth0 was connected to internet via router, it is highly likely that it was exposed to a larger set of attack vector as compared to wlan0 so it requires comparatively larger set of Snort detection rules. One quick method is to enable

all rules provided by Snort, Second method is to choose adoptive approach and select environment specific rules only. For instance, no POP3 server was running on RPi gateway so no POP3 rule were required.

Only those rules were enabled which are effective in detection of major attacks against IoTs, e.g. rules related to DoS, botnets, scanners, telnet, SSH brute-forcing(dictionary attacks) malware patterns and few other. An approx of five thousand rules were selected for eth0. Some of the sample Eth0 rules are presented in Appendix G.

4.3.2 Snort Instance Wlan0

Similar to eth0, interface wlan0 was loaded with separate Snort instance. Listing 4.2 shows the configurations for config file snort_wlan0.conf.

```
#copy original snort.conf file
Sudo cp /etc/snort/snort.conf snort_wlan0.conf
sudo nano /etc/snort/snort_wlan0.conf
#configurations start from here
ipvar HOME_NET 192.168.42.0/24
ipvar EXTRNAL_NET !$HOME_NET
var RULE_PATH /etc/snort/rules
var SO_RULE_PATH /etc/snort/so_rules
var PREPROC_RULE_PATH /etc/snort/preproc_rules
var WHITE_LIST_PATH /etc/snort/rules
var BLACK_LIST_PATH /etc/snort/rules
config pcre_match_limit: 3500
config pcre_match_limit_recursion: 1500
config detection: search-method lowmem search-optimize max-pattern-len 20
preprocessor perfmonitor:
    time 300 events atexitonly max console pktcnt 1000
output log_tcpdump: /var/log/snort/eth0/snort.log
include $RULE_PATH/wlan0.rules
```

Listing 4.2: snort_wlan0.conf

Unlike eth0 instance which was using ac-bnfa-nq pattern matcher algorithm, instance wlan0 was configured with lowmem algorithm. As it's name indicates, it is a medium performance and low memory algorithm, which matches wlan0 requirements.

4.3.2.1 Network

Interface wlan0 was running it's own isc-dhcp-server with network 192.168.42.0/24 and subnet 255.255.255.0. Since wlan0 was utilized as access point (AP) to provide wireless connectivity to sensors, all the sensors obtained ip addresses from 192.168.42.0/24 ip-pool. In order to turn wlan0 WiFi into an AP, an open

source package *hostapd* was used. Appendix B shows configuration process of isc-dhcp-server and hostapd on RPi.

4.3.2.2 Wlan0 Rules

Interface wlan0 was least susceptible to a larger attack vector, the reason is that it is less exposed to direct attacks on AP. If an attacks comes through the internet i.e. through interface eth0, Snort eth0 instance will already detect it. Some of the direct attacks on AP might not be detected by Snort because it is not able to sniff or commence the packets. For instance, evil twin and de-authentication attacks can not detected by Snort, to detect such attacks other tools like Kismet[72] can be used, which is out of the scope of this thesis project.

Considering above discussed phenomenon, a relatively small set of rules was selected for wlan0 Snort instance. A sample rule can be as follows:

```
#rule for botnet C2C flow
alert tcp any any -> any 23 (msg:"Mirai C2C flow, inbound login attempt";
    flow:to_server,established; content:"xc3511";depth:6; sid:1000000;
    rev:1)
```

4.3.3 Logging

Both of the Snort instances were set to log to their own logging directories, i.e. instance eth0 logs to directory /var/log/snort/eth0/ while instance wlan0 logs to /var/log/snort/wlan0 directory. These loggs were shipped to ELK server using Beats data shippers. Following section provides details of how Beats were configured on RPi.

4.4 Configuring Beats on RPi

Filebeat and Metricbeat are the important component of Beats package. They include internal modules that simplify collecting, parsing and visualizing common log formats such as, NGINX, Apache and system metrics. It is recommended not to install Go, which is the language Beats are written in, by apt-get which at the time of writing this thesis, does not compile on RPi with Raspbian stretch and generates errors.

A stable Arm release of Go was installed from official Go repository[73]. Appendix C shows the process of installing Go and Beats on Rpi. To setup Filebeat and enable system module, *./filebeat modules system enable* was used, while to ship data to ELK server, *./filebeat -e* was used. Similar is the process of using Metricbeat.

4.5 Configuring ELk server

ELK server was setup on Openstack cloud where it was enabled to receive logs and RPi performance data including, CPU, memory, system load and network information. Kibana dashboard can be access remotely by *https://ElK-Server-IP* and providing username and password. It is important to open port 5044 on Openstack, it can be done by: Access & security > Manage Rules > Add Rule > port 5044. Appendix D provides script of how to install and configure all the necessary components of ELK stack as well as Logstash filters for parsing Snort alerts. It is recommended to install older stable JDK version (Java version 8.1 is a good option), as by date of this thesis writing, Logstash version 6.3 was not compatible with latest Java distribution.

4.6 Testing

To evaluate the performance of RPi and Snort, different rates of syn-flood attacks in terms of MBit/sec were initiated from IDS testing machine with Kali. Test Data rates were 100MBit/sec, 300MBit/sec, and 500MBit/sec. A bash script was written and run on RPi which can automatically start both of the Snort instances at interfaces eth0 and wlan0 and stop them after time interval of 600 seconds. The script can start Filebeat and Metricbeat as well in the same time-frame, same script also monitored RPi resource consumption and generated the graph depicting RPi's CPU and memory status during the attacks.

An another script was started on IDS test machine (attack machine) which initiated syn-flood attacks at different data rates for each test individually. The purpose of testing was to accomplish objective *(b)* of problem statement, i.e. *Does RPi have enough resources to support Snort as an IDS, in terms of resource consumption?* In order to test RPi capability of hosting Snort as an IDS, interface eth0 was tested first at multiple attack data rates. Interface eth0 was attacked by syn-flood while wlan0 was attacked by ssh-dictionary attack (ssh brute-force) from one of the compromised sensors (another RPi used as sensor) connected to RPi's access point.

Interface eth0 with Snort instance_eth0 was tested individually, in final test, both of the interfaces were attacked at the same time to test the worst case scenario. Figure 4.2 shows the attack points on RPi gateway, as well as the Snort instances responsible to detect these attacks.

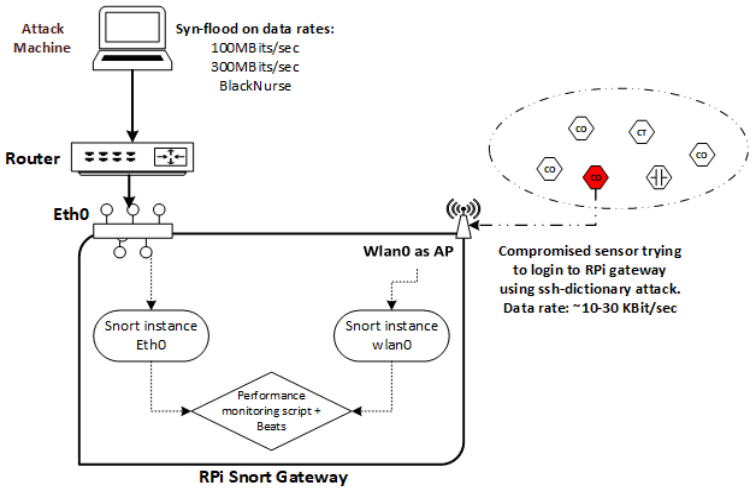

Figure 4.2: *Figure created from system design figure, it shows attack vectors on each interface*

4.6.1 Test 1

The first test was to syn-flood interface eth0 at 100MBit/sec data rate from attack machine. Interface eth0 of RPi can facilitate a maximum of 300MBit/sec of bandwidth. The purpose of this test was to evaluate the performance of RPi while it is under attack and Snort IDS and Beats are running. Table 4.1 shows the characteristics of Test 1.

Parameter	Value
Attack Data Rate	100MBit/sec
Attack Type	Syn-flood
Number of Rules	≈7000
Interface under Attack	Eth0
Run Time	600 Seconds
Instances	Eth0, wlan0 and Beats

Table 4.1: *Characteristics of Test 1, syn-flood at 100MBit/sec*

Test 1 was run for 600 seconds with a total rule set of 7000 rules, where 6000 rules were assigned to Snort instance eth0 and an approx of 1000 rules were provided to wlan0. It is important to note that Snort instance wlan0 was running but was not under attack in any next tests except test 4. To load Snort instances and Beats, the script in appendix E was run with *bash snort.sh* while at the same time syn-flood attack from attack machine was initiated by hping3 command:

```
hping3 -d 0 -S -p 80 --flood --rand-source 192.168.10.20
```

The hping3 flooded the interface eth0 of RPi with packets set with syn flags (-S option) with random source ip addresses. *-d 0* option sets data payload to 0 while option *-p 80* sets the target port to port 80. This Command can generate an approx data rate of ≈100MBit/sec.

4.6.2 Test 2

Test 2 differs from Test 1 by a change in attack data rate from attack machine. For test 2, attack data rate was set to 300MBit/sec. The purpose of this attack was to analyze full capability of interface eth0 and make RPi CPU to work on more packets. i.e. interface eth0 responds with an *ack* to each *syn* packet, resulting in an increased CPU consumption, also, Snort has to commence and process more packets per seconds which also increases CPU usage. Test 2 characteristics are provided in Table 4.2.

Parameter	**Value**
Attack Data Rate	300MBit/sec
Attack Type	Syn-flood
Number of Rules	≈7000
Interface under Attack	Eth0
Run Time	600 Seconds
Instances	Eth0, wlan0 and Beats

Table 4.2: *Test 2 characteristics and it's values for a higher data rate of 300MBit/sec*

The hping3 command used to generate syn flood on 300MBit/sec data rate is given as:

```
hping3 -d 120 -S -p 80 --flood --rand-source 192.168.10.20
```

All the other parameters of above hping3 command remained the same as test 1, except the packet payload size i.e. option *-d 120*, which makes hping3 to send each syn packet with a packet payload size of 120 bytes. Hping3 not only increases the packet size but also increases the packets/sec rate to maintain the maximum flood rate. Test 2 sent syn packets at the rate of ≈300MBit/sec which matches the maximum supported rate of Ethernet interface eth0 of RPi.

4.6.3 Test 3

Test 3 was meant to increase the CPU clock rate of RPi by flooding ICMP type 3 code 3 packets, destination unreachable and port unreachable respectively. This type of attack is referred as BlackNurse. This attack is notorious for very high CPU consumption even at low rate of only ≈130MBit/sec. There are several systems which are vulnerable to this kind of attack. By using even a single laptop, this attack can shutdown a vulnerable server by over-loading the CPU. Table 4.3 gives details of test 3 characteristics in the following:

Parameter	**Value**
Attack Data Rate	130MBit/sec
Attack Type	BlackNurse
Number of Rules	≈7000
Interface under Attack	Eth0
Run Time	600 Seconds
Instances	Eth0, wlan0 and Beats

Table 4.3: *Test 3 characteristics while it used BlackNurse attack at relatively low data rate*

Following command can be used to initiate this attack:

```
hping3 -1 -C 3 -K 3 --flood --rand-source 192.168.10.20
```

In the command above, switch *-1* sets attack mode to ICMP, *-C 3* selects code 3 of ICMP (destination unreachable), *-k 3* sets type of ICMP to type 3 (port unreachable).

4.6.4 Test 4

Test 4 attacked both of the interfaces at the same time, i.e. interface eth0 and wireless interface wlan0. The purpose of this test was to evaluate the worst case scenario of RPi as an IDS machine. Worst case scenario in the sense that it is less likely that both of the interfaces could be attacked at the same time. This was the test where both of the Snort instances were detecting the attacks, generating alerts and writing logs to their respective logging directories.

From attacker machine, same syn-flood attack was initiated against eth0 at the data rate of 300MBit/sec, while interface wlan0 was attacked by a compromised sensor already connected to RPi's access point. For interface wlan0 attack, it was assumed that the compromised sensor (another RPi) was trying to login to the RPi gateway through ssh brute-force (Dictionary attack). The Dictionary attack is a low rate attack which requires data rate of ≈10-30KBit/sec.

The ssh dictionary attack uses a dictionary of usernames and passwords to login to a system. If any username and password matches, attack is successful. The aim to choose this attack was to simulate the real life behaviour of most of the botnet command and control servers, which use the same attack methods to compromise machines with weak/default usernames and passwords. Table 4.4 presents the details of attack characteristics selected for test 4.

Parameter	**Value for Eth0**	**Value for Wlan0**
Attack Data Rate	300MBit/sec	30KBit/sec
Attack Type	Syn-flood	ssh bruteforce
Number of Rules	≈6000	≈1000
Interface under Attack	Eth0	Wlan0
Run Time	600 Seconds	600 Seconds
Instances	Eth0, Wlan0 and Beats	Eth0, Wlan0 and Beats

Table 4.4: *Test 4 characteristics. It shows a dual attack on both interfaces with different data rates.*

The hping3 command was used from attacker machine, while nrack was used from compromised sensor to initiate ssh bruteforce. The attack commands are given as follows:

```
ncrack -p 22 --user root -P dark-web-top-10000.txt -T5 192.168.42.1
hping3 -d 120 -S -p 80 --flood --rand-source 192.168.10.20
```

The hping3 command is same as in test 2. while in ncrack, option *-p 22* sets target port to port 22 (ssh port), option *–user root* sets target username to be probed to root, option *-P* loads the file containing a dictionary of passwords, dark-web-top-10000.txt file in this case, option *-T5* increases the prob rate to 5 tries in a single prob, 192.168.42.1 is the target ip address i.e. gateway ip of wlan0 access point. Test 4 was very important because it tested the decision whether it was sensible to run two Snort instances on a low spec device Raspberry Pi, also test 4 pushed the RPi capability to it's limits. Results obtained from each test as well as response of Beats and monitoring server are discussed in next chapter i.e. chapter 5 *Analysis*.

Chapter 5

Results and Analysis

This chapter provides an analysis of all the conducted tests of Raspberry Pi and Snort as an IDS. This chapter evaluates the results in connection to one of the core objectives of this thesis, i.e. *Does RPi have enough resources to support Snort as an IDS, in terms of resource consumption?*

5.1 Test 1

Test 1 was taken to evaluate the RPi CPU and memory utilization during a relatively low rate syn-flood attack on 100MBit/sec. The focal point of all the tests was CPU consumption while memory was an added parameter because main memory consumer is the Snort rule set which stayed constant during all the tests. The reason to add memory parameter into tests was to give an estimate that how much Snort can influence the RPi memory for a given set of rules.

It is currently not possible to receive Snort overall statistics while Snort is running, the only way to get Snort statistics is to read Snort I/O summary after it exits, therefore, Snort data was obtained when Snort stopped after the test i.e. precisely after 600 seconds. Snort was able to commence 3037094 packets, where it analyzed a total of 225408 packets, marking success rate in terms of packet analysis to be 7.42%. Snort dropped almost half of the packets i.e. 2811684 with a percentage of 48.07%, it is worth mentioning that packets dropped by Snort should not be confused by the packets dropped by Ethernet interface, in fact, packets dropped by Snort are the packets which are already passed by the interface and commenced by Snort but Snort could not inspect them from the memory. Table 5.1 shows the details of Snort I/O values during the Test 1.

The largest portion of the packets were outstanding packets i.e. 2811686 with a percentage of 92.578%. Outstanding packets are those packets which

Snort I/O	Value
Packets Received	3037094
Packets Analyzed	225408 (7.42%)
Packets Dropped	2811684 (48.07%)
Outstanding Packets	2811686 (92.578%)
Packets/min	25045

Table 5.1: Test 1 - *Snort I/O Statistics, showing Snort analysis and packet drop rate. Percentage values are rounded.*

are loaded into memory but are either marked as dropped or are still waiting for Snort detection engine to be processed. Snort packet drop depends on the incoming data rate, packet size and CPU load.

A high data rate increases the CPU consumption. Figure 5.1 shows the CPU and memory consumption of RPi during test 1. It can be seen in Figure 5.1 that memory consumption is almost static, approximately 36% where it is being loaded with Snort, Beats, as well as RPi's own processes.

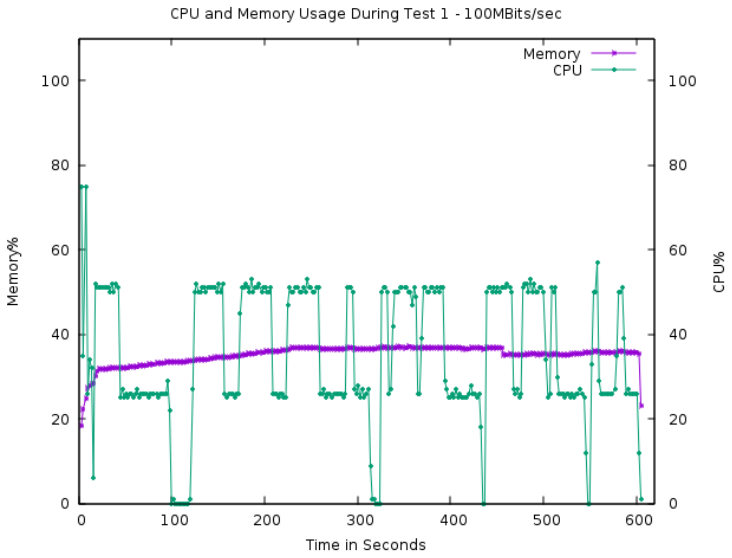

Figure 5.1: Test 1 - *Memory and CPU consumption during test 1 at 100MBit/sec data rate.*

Such a low memory consumption is encouraging to add more Snort rules but it must be taken into consideration that introducing new rules can also increase CPU consumption because Snort will have to process more rules to match. In terms of CPU consumption, test 1 at 100MBit/sec clocked the CPU to an

average of 56% of total CPU, at this percentage, RPi was able to perform it's operations effortlessly without any performance lagging. During test 1, it had been noticed that Metricbeat and Filebeat kept connecting and disconnecting to ELK server periodically, showing that on RPi, at even 100MBit/sec attack data rate, some of the services can be disrupted, although ssh kept operational.

In Figure 5.1, certain fluctuations can be seen in CPU usage, one of the reasons is that in test one, packet size was small with no data payload and Snort kept refreshing it's packet buffer whenever the buffer got overloaded, with each buffer refresh, CPU usage dropped and then it gradually got higher til next buffer reload, this cycle continuous until Snort commences different size of packets. It is important to note that buffer refresh does not follow any particular time interval.

5.2 Test 2

Test 2 was conducted at higher attack data rate of 300MBit/sec with a larger packet size of 120 bytes of data payload. Purpose of this attack was to evaluate RPi response to a higher data rate syn-flood attack in terms of resource consumption, as well as to evaluate Snort performance. It is worth mentioning that RPi model 3 B+ Ethernet interface can support a maximum of 300Mbps of throughput, any data rate above that will be bottlenecked at eth0.

Snort was expected to commence more packets with a higher drop rate. Table 5.1 shows the statistics of Snort after Test 2 was finished. Snort was able to commence 4590679 packets, which is 33.8% higher than test 1.

Snort I/O	**Value**
Packets Received	4590679
Packets Analyzed	245358 (5.35%)
Packets Dropped	4341882 (48.61%)
Outstanding Packets	4345321 (94.65%)
Packets/min	27262

Table 5.2: Test 2 - *Snort I/O Statistics, showing Snort analysis and packet drop rate. Analysis rate has dropped while outstanding rate has increased.*

Although it was expected that Snort will be commencing more packets in test 2 but Snort did not go through each and every packet, most of the traffic was passed and not analyzed which Snort does not count in I/O counters. Snort dropped 48.61% of packets which is not much different than test 1. Snort was able to analyze only 5.35% of packets with 94.65% of outstanding rate.

Since packet size was larger in test 2, Snort spent more time on inspecting packets which raised outstanding packet rate and dropped the packet analysis

rate, such behaviour of Snort should also be evident in terms of CPU consumption, i.e. it should be more consistent with less spontaneous fluctuations because there is less frequent buffer reloads by Snort for larger packets, as discussed in test 1. Figure 5.2 confirms this behaviour of Snort where CPU fluctuates very close to the average CPU consumption of 64.7% with less frequent changes.

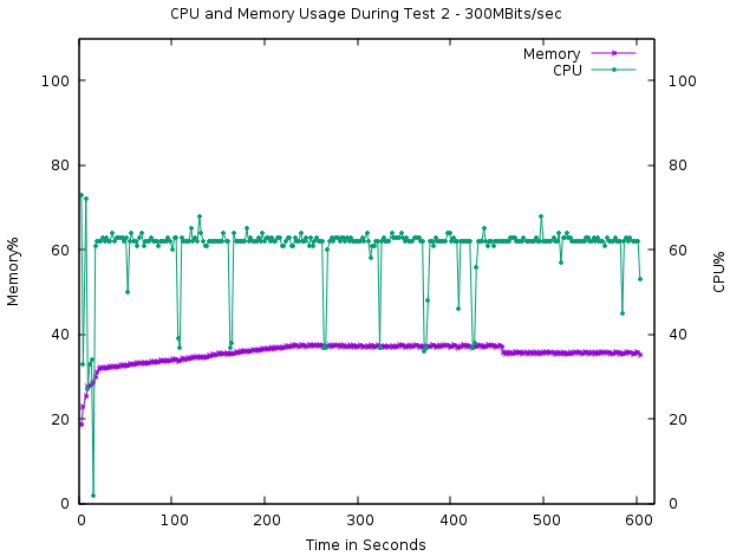

Figure 5.2: Test 2 - *Memory and CPU consumption during test 2 at 300MBit/sec data rate. CPU consumption is increased but consistent as compared to test 1.*

There was a ≈15% increase in CPU consumption as compared to test 1, which was not as higher as it was expected according to data rate ratio. The memory consumption was almost the same as test 1 because no new Snort rules were added during test 2. A different attack type might increase or decrease memory based on it's rule class and number of associated rules. Test 3 will explain this phenomenon.

During test 2, it was noticed that RPi's monitoring service was down, both of the Beats were unable to connect to ELK server, unless the attack was stopped. It shows that a 300MBit/sec syn-flood attack can knock of most of RPi's Ethernet associated services i.e. ssh was down and it was not possible to connect to RPi via ssh. Although with a 64.7% CPU usage, RPi was able to run it's internal services smoothly and no performance lag was noticed, RPi as an IDS was able to perform fine during test 2 but without remote monitoring, which makes it less significant as a remotely operated IDS system.

During test 2, it was concluded that Ethernet interface of RPi with only 300Mbps throughput support, can lead to failure of RPi as a remote IDS.

5.3 Test 3

Test 3 changed the attack type from syn-flood to ICMP flood attack called BlackNurse. This attack was meant to assess RPi's vulnerability to this attack and to analyze CPU consumption. During test 3, Snort was able to detect this attack and generate alerts. Compared to syn packets, Snort handles ICMP packets more smoothly with higher analysis rate and comparatively lower drop rate. Table 5.3 shows the results summary obtained from Snort after the attack.

Snort I/O	**Value**
Packets Received	16675421
Packets Analyzed	3532178 (21.18%)
Packets Dropped	13131500 (44.05%)
Outstanding Packets	13143243 (78.82%)
Packets/min	392464

Table 5.3: Test 3 - *Snort I/O Statistics, showing Snort analysis and packet drop rate during BlackNurse attack. A higher packet commence rate by Snort can be noticed.*

It can be noticed in Table 5.3 that Snort was able to receive much more packets than syn-flood, it is because Snort's traffic pass rate is lower for ICMP traffic, Snort captures most of the ICMP packets and lets pass less ICMP traffic as compared to syn packets in previous two tests.

Snort received 16675421 packets with packet analysis rate of 21.18% i.e. a total of 3532178 packets. Drop rate of Snort was lower in test 3 as compared to previous tests, 13131500 packets with a percentage of 44.05%. Outstanding packets were 78.82% of total packets, i.e. 13143243 packets were outstanding packets which were either marked to be dropped or were waiting for being analyzed. For test 3, Snort performance seems reasonably acceptable.

As mentioned above, BlackNurse attack can result in very high CPU usage. Figure 5.3 is evident that during test 3, RPi's CPU usage clocked to nearly 100%, with an average CPU usage of 96%. Even at this CPU rate, RPi was able to perform it's IDS functions smoothly, no performance lags or system shutdown was noted during the attack. Also, all the services associated with Ethernet were operational, except few disconnects of Beats from ELK server but as overall, performance of RPi during test 3 was satisfactory. Test 3 also confirms that RPi model 3 B+ is not vulnerable to BlackNurse although it did increase it's CPU but could not paralyzes it completely.

In Figure 5.3, a slight decrease in memory can be noted, it is because Black-Nurse attack belongs to different attack class with only single detection rule as compared to syn-flood which has several rules to be loaded into memory and matched. Test 3 concluded that RPi can perform as an IDS device even at an

average CPU usage of 96%.

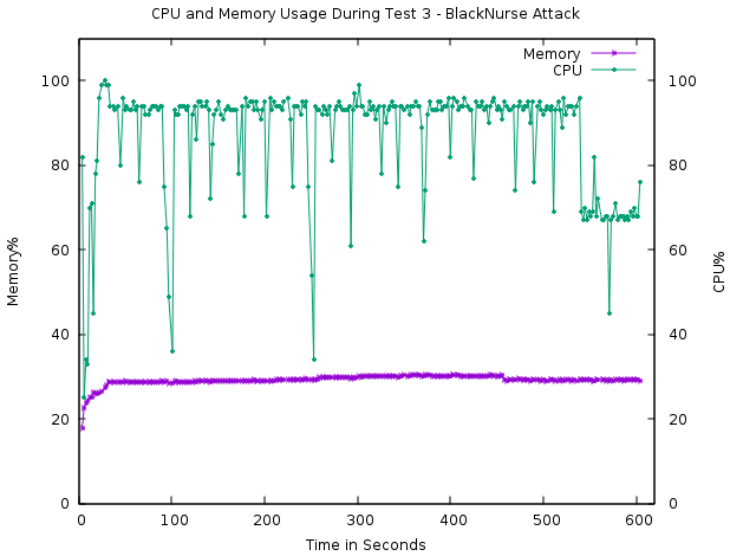

Figure 5.3: Test 3 - *Memory and CPU consumption during BlackBurse attack, a very high CPU rate can be noticed with slight decrease in memory consumption.*

5.4 Test 4

Test 4 was conducted to test the worst case attack scenario for RPi and Snort. In this test, both of the interfaces i.e eth0 and wlan0 of RPi were attacked with different type of attacks. Interface eth0 was syn-flooded while wlan0 was under ssh dictionary attack from a connected compromised sensor. It is worth mentioning that generally sensors do not have the capability for a ssh bruteforce except those IoT devices which have the capability of using for instance, Telnet. The attack scenario was conducted to evaluate performance of both Snort instances and performance of RPi under dual attack.

Table 5.4 and Table 5.5 show statistics of Snort instance eth0 and Snort instance wlan0 respectively.

Snort I/O	Value
Packets Received	6604904
Packets Analyzed	77635 (1.175%)
Packets Dropped	6519573 (49.675%)
Outstanding Packets	6527269 (98.825%)

Table 5.4: Test 4 - *Snort I/O Statistics for eth0, showing Snort statistics during dual interface attack. A very low analysis rate can be seen, with a very high outstanding packet rate.*

Snort I/O	Value
Packets Received	41902
Packets Analyzed	41796 (99.75%)
Packets Dropped	0 (0.0%)
Outstanding Packets	106 (0.25%)

Table 5.5: Test 4 - *Snort I/O Statistics for wlan0, showing Snort statistics during dual interface attack. A very higher packet commence rate by Snort can be noticed with no packet drop.*

Snort instance eth0 was dropped to an analysis rate of only 1.175% because almost half of the buffer was taken over by Snort instance wlan0 as well as the CPU. A larger portion of outstanding packets, 6527269 with a percentage 98.825% kept waiting for Snort to get analyzed or got dropped eventually after time out. Out of these outstanding packets, 49.675% i.e. 6519573 packets got timed out and dropped. During test 4, Snort instance eth0 performance was below satisfactory but Snort was still able to detect the attack and generate alerts.

Snort instance wlan0 was under a low rate attack of ≈30KBit/sec in test 4. During the test, wlan0 instance received 41902 packets, out of which it analyzed 41796 packet with a analysis rate of 99.75%. None of the packets were dropped by wlan0 instance while only 106 packets, 0.25% were in the outstanding waiting list which shows that Snort can perform very good while dealing with low rate attacks.

It was expected that putting both of Snort instances to work at the same time will increase CPU to it's maximum. Dual attack clocked the CPU to it's full capacity i.e. ≈100% with an average CPU usage of 98.7%. This rate of CPU does not let Snort detection engine to work smoothly while dealing with high data rate attacks, resulting in much higher outstanding and drop rate, as well as a low analysis rate.

During this test, almost with full CPU utilization, RPi was still responsive and no service disruption or system shutdowns were noticed. While services

related to Ethernet interface eth0 i.e. ssh and Beats were facing multiple service disconnects, as like in test 2. On interface wlan0, access point was still functioning and it was possible to connect to RPi's access point. Much like test 1 and test 2, no major change in memory consumption was noticed.

A dual attack of RPi is less probable, as it will be rare that both of the interfaces will be attacked at the same time, but this test does show that maintaining this attack for a long time can probably overwhelm the CPU and can result in a complete service disrupt. This test concluded that RPi as an IDS gives below average performance in terms of detection of dual interface attacks and CPU consumption. Also, remote monitoring system tends to get disrupted easily because of the RPi's 300MBit Ethernet port. Figure 5.4 shows the results obtained after test 4, a very high CPU usage can be seen.

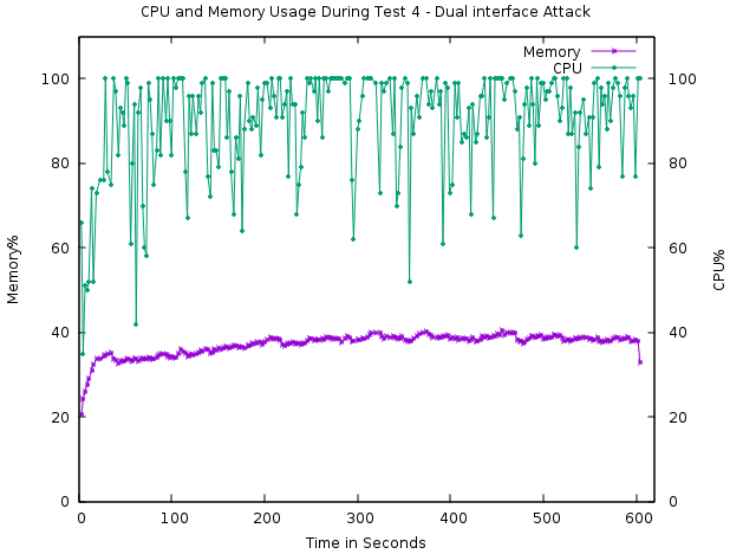

Figure 5.4: Test 4 - *Memory and CPU consumption during dual attack, a very high CPU rate can be seen because both of the Snort instances are operational in detecting attacks.*

5.5 Monitoring Server

The main purpose of including monitoring service was to visualize the Snort data and to provide simple readability to Snort alerts e.g. type of active attacks, number of alerts generated by Snort, location of attacking IP addresses and details of the Snort rule which detected the attack i.e Signature ID etc. Beats components, *Filebeat* was responsible for reading Snort alerts at RPi and shipping them to ELK server.

Another Beats component called *Metricbeat*, was used to ship RPi's system metrics to ELK server. It is worth mentioning that for calculation of RPi's CPU and memory during the tests, local script was preferred instead Metricbeat, this step was taken to evade time delay and service disconnect issues of Metricbeat.

During the time period of all the tests, both of the Beats were running alongside the Snort instances. Except test 1 and test 3, Beats lost their connection to ELK server because Ethernet interface eth0 was under heavy load and was unable to support any more traffic. When the attacks were stopped, Beasts reestablished their connections to ELK server. Filebeat uses the actual timestamp of alert logs after reconnect but Metricbeat sometime gives less precise results in relation to timestamp. In following section, data received by Beats on the ELk server is presented.

5.5.1 Filebeat

Filebeat was configured to ship Snort alerts only and no other logs were sent through it. It was more resilient than Metricbeat in terms of having less frequent disconnects during the attacks. In the following section, Snort alert messages sent by Filebeat for different attacks are shown.

Figure 5.5 presents the syn-flood attack on eth0, detected by Snort. It is harder to read Snort alerts on RPi itself, ELK presents the alert in an easy, human readable form.

t	message	[**] [1:10002:1] TCP SYN flood attack on Port 80 detected [**] [Classification: Attempted Information Leak] [Priority: 2] 12/11-18:38:55.939300 24.118.174.36:7747 -> 192.168.10.20:80 TCP TTL:63 TOS:0x0 ID:1414 IpLen:20 DgmLen:260 ******S* Seq: 0x17C7ACB1 Ack: 0x3945567 Win: 0x200 TcpLen: 20
t	options	DgmLen:260 ******S* Seq: 0x17C7ACB1 Ack: 0x3945567 Win: 0x200 TcpLen: 20
#	priority	2

Figure 5.5: Syn-food attack on eth0 - *Snort detects the attack and generates alert, which is sent by Filebeat and processed by ELk index patterns.*

Figure 5.6 shows the detected ssh brute-force attack landed against wlan0 from compromised sensor. It shows the alert message, priority of attack, source ip of the attacker and signature id of the detection rule.

t	message	[**] [1:19559:5] INDICATOR-SCAN SSHbrute force login attempt [**] [Classification: Misc activity] [Priority: 3] 12/05-15:20:08.722321 192.168.42.30:54184 -> 192.168.42.1:22 TCP TTL:64 TOS:0x0 ID:10258 IpLen:20 DgmLen:1520 DF ***A**** Seq: 0xDEFA4C7 Ack: 0x70009034 Win: 0x7C80 TcpLen: 32
t	options	DgmLen:1520 DF ***A**** Seq: 0xDEFA4C7 Ack: 0x70009034 Win: 0x7C80 TcpLen: 32
#	priority	3

Figure 5.6: ssh-bruteforce attack on wlan0 - *It shows the generated alert by Snort after detection of the attack.*

similarly, Figure 5.7 and Figure 5.8 show the detected Snort alerts, presenting alert message of blackNurse attack and XMAS scan respectively.

t	message	[**] [1:88000012:1] TDC-SOC - Possible BlackNurseattack from external source [**] [Priority: 4] 12/12-15:03:17.520920 192.168.20.100 -> 192.168.10.20 ICMP TTL:63 TOS:0x0 ID:5017 IpLen:20 DgmLen:56 Type:3 Code:3 DESTINATION UNREACHABLE: PORT UNREACHABLE ** ORIGINAL DATAGRAM DUMP: 1.2.3.4:0 -> 5.6.7.8:0 TCP TTL:64 TOS:0x0 ID:4783 IpLen:20 DgmLen:28 Seq: 0x8EFC1 ** END OF DUMP [Xref => http://soc.tdc.dk/blacknurse/blacknurse.pdf]

Figure 5.7: blackNurse attack on eth0 - *Snort successfully detects the attack and generates alerts. Figure shows the alert message on the ELK server.*

t	message	[**] [1:1228:7] SCAN nmap XMAS [**] [Classification: Attempted Information Leak] [Priority: 2] 11/18-13:43:56.275071 192.168.20.100:63165 -> 192.168.10.20:5859 TCP TTL:49 TOS:0x0 ID:14560 IpLen:20 DgmLen:40 **U*P**F Seq: 0x894F8E1A Ack: 0x0 Win: 0x400 TcpLen: 20 UrgPtr: 0x0 [Xref => http://www.whitehats.com/info/IDS30]
#	offset	682,774
t	options	DgmLen:40 **U*P**F Seq: 0x894F8E1A Ack: 0x0 Win: 0x400 TcpLen: 20 UrgPtr: 0x0 [Xref => http://www.whitehats.com/info/IDS30]
#	priority	2

Figure 5.8: XMAS scan detected on eth0 - *Figure shows the detection of nmap XMAS scan, which normally belongs to first phases of a hacking attempt.*

For a system administrator, every information is vital for both pre and post attack countermeasures. Specifically, it is very useful if location of attack IPs could be identified, because the admin can then blacklist those ips either based on subnet or location. ELK server also provides geoip map of all the unique ip addresses used for attacks. Figure 5.9 shows the map of countries where the attacks were coming from. The map was generated from test 1 syn flood attack where interface eth0 was flooded with spoofed random source addresses, just to generate different ip data.

It must be taken into consideration that source ip address may not be the

actual ip of the attacker in certain type of attacks because hackers generally spoof their ip addresses (if packet response is not required) or use proxies. The GeoIP is very useful against scanners, bots traffic, and web crawlers.

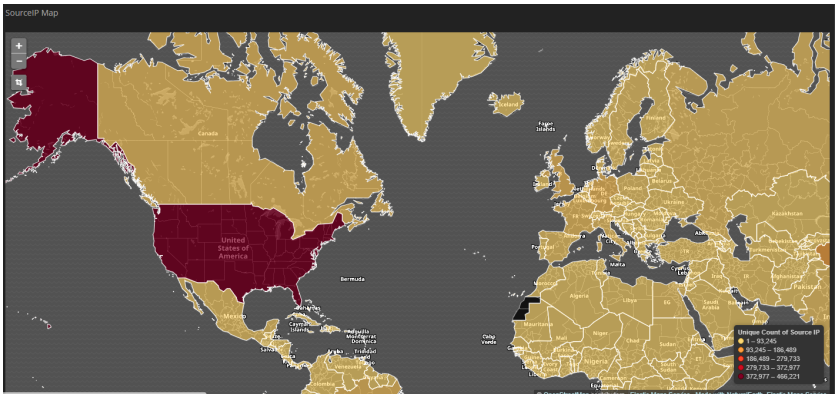

Figure 5.9: GeoIP MAP - *Showing the country location of attackers based on their IP addresses. Red color countries are the top countries where attacks where generated from.*

5.5.2 Metricbeat

Metricbeat suffered most of the service disconnects from ELK server as compared to Filebeat. It was an additional Beats component added to monitor RPi's metrics. It can be very useful in cases where any unexpected behaviour of a system is to be monitored, for example, CPU or memory usage above certain threshold, level of incoming traffic etc. As discussed before that Metricbeat might give errors with correct timestamps while shipping data, it was hard to generate graphs with precise time of attacks, instead, graphs with a longer time frames are shown in following section. Figures 5.10 and 5.11 show the system load, memory usage and number of incoming and dropped packets respectively during a time span of 30 days.

In figure 5.10, system load gives information of average system load of RPi in 15 minutes interval in terms of percentage. Similarly, memory represents the average memory usage of RPi in term of MBs during same time interval. It must be noted that resource consumption statistics generated by local script on RPi may not correlate or synchronize with Metricbeat results due to service disconnect issues of Metricbeat.

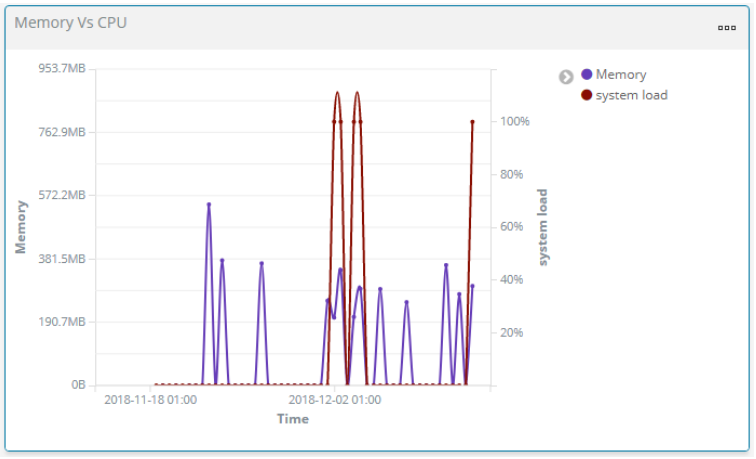

Figure 5.10: Memory and system load - *The line graph generated at ELK server using Metricbeat data, showing memory and system load fluctuations during a longer time span.*

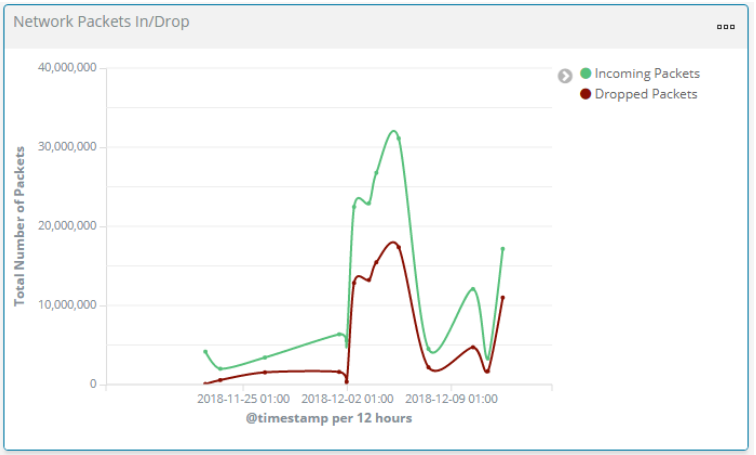

Figure 5.11: Incoming and Dropped packets - *The number of packets received and dropped by Ethernet interface. Based on Data shipped by Metricbeat.*

Figure 5.11 gives an insight into level of traffic RPi was exposed to during a total time interval of 30 days. It gives a comparative analysis of number of incoming packets on interface eth0 and number of packet that were dropped by the same interface. It can be seen that RPi dropped a large number of packets which clearly show a major weakness of RPi's 300Mbps Ethernet interface. When this interface is exposed to above 300MBit/sec traffic, it starts dropping large amount of packets which may or may not effect the RPi CPU depending

on the nature of packets. For instance, mere syn packets have a very less effects on CPU in normal situations but ICMP type 3, code 3 packets demand a large CPU utilization even on comparatively low rate traffic.

Chapter 6

Discussion

This chapter provides a critical discussion on project evaluation, the adopted approach, implementation, results and analysis and on the project as a whole. Also, suggestions on how this project can be improved is presented in future work.

6.1 Project Evaluation

IoT market space is expected to be massive in near future. IoT devices are expected to have direct impact on businesses and everyday lives of people. With this tremendous growth of IoTs, unfortunately, attacks on these devices has increased many fold because of weak to no security of IoT devices. Most of the IoT manufacturers focus on higher profit margins, overlooking security aspects. Most of the commercial security products, specifically, intrusion detection devices (IDS) are too costly and demand special hardware, which makes it difficult for average IoT user e.g. smart home or smart building, to afford these IDS devices.

Keep in view above phenomenon, this thesis was intended to build and test an affordable IDS for common IoT environments. The project used a low price, 35$ single board computer device Raspberry Pi and open source IDS Snort.

6.2 The Approach

Snort IDS was placed in a single centralized RPi gateway for simplicity. For testing Snort and RPi as an IDS, only those attacks were chosen which are very common against IoTs i.e. DoS, dictionary attacks and port scanners. The project chose an empirical approach and used actual hardware instead simulation,

also, real attack tools like hping3, ncrack and nmap were used for testing of IDS.

Two instances of Snort were deployed on RPi, one for each interface, i.e. Ethernet interface eth0 and wireless interface wlan0. The reason for using dual Snort instance was because Snort is single threaded and can not listen on more than one instance at a time. Although this approach can be resource overhead if no attacks on wlan0 are expected. Also, certain types of attacks on wlan0 can not be detected by Snort, for instance, wireless de-authentication attack on wlan0. Both of the instances were provided with different number and type of rules based on the nature of their exposure, a total of ≈7000 rules were used.

Two attack points were chosen on RPi, Ethernet interface was attacked using different data rate attacks, wlan0 was attacked by a compromised connected sensor with a ssh dictionary attack. To analyze the Snort alerts and visualize Rpi metrics, Beats were used alongside remote monitoring server equipped with ELK stack.

6.3 Implementation and challenges

At the time of this thesis writing, only few documented implementation of Snort with RPi can be seen in reputable journals. It was hard to obtain closely related research work which used the same setup. in fact, no matching setup was found till date which uses a combination of Snort, RPi and ELk stack, this project was the first which attempted this combination.

Rpi uses Armv Linux architecture, which has several dependency issues while compiling common Linux packages. Compilation of Snort on RPi was not straight forward and had to be compiled and installed manually form source. It was done after several trial and error and solving dependency issues. Also, choosing *ac-bnfa-nq* algorithm over Snort default algorithm *ac-split* was not an instant decision. unfortunately, no reputable document exists that could provide a comparative performance analysis of Snort's current pattern matcher algorithms. To seek professional advice, Snort developers were consulted on *seclists.org* which is an open portal for Snort and other related security software.

Selecting and writing Snort rules was also a difficult process because a single bad written rule could give inaccurate test result, for instance, in a syn-flood attack rule, if *count 1000, seconds 5* is changed to *count 10, seconds 5*, it will not only generate a very large amount of alerts but also increase CPU many folds. Similarly, if *flags:S* was put at the end of rule instead in beginning, it can demand more work from pattern matcher algorithms hence increasing resource usage. Moreover, selection of individual rules which match this project criteria was a worth a job.

Configuring Beats components for data shipping on RPi was the most time

intensive. Currently, *Elastic.co* which provides official support for Beats and ELk stack, do not provide Arm architecture builds of Beats. Also, Beats use Go language for compilation, which also demands a manual labour to install and configure.

The project first planned to use rsyslog for sending Snort logs to Logstash but rsyslog did not provide support to ship real time system metrics of RPi. Beats, on the other hand, could not only send Snort alert logs but also could send real time system metrics. It was more reasonable to use Beats as it is easier to index data on ELK stack when using Beats (Except Snort index pattern for Snort alerts which had to be written). Installing ELK stack on remote server which is an Ubuntu machine, was comparatively easier and straight forward. This chapter also provided the answer to first question of problem statement i.e *How can we build an affordable and portable IDS solution using RPi and Snort?*

6.4 Testing and Analysis

Two testing parameters were set, first, to evaluate Snort performance in terms of detection rate i.e. analysis rate, packets dropped, and number of outstanding packets, problem with these Snort statistics was that Snort does not provide live on the run statistics, these can only be obtained after Snort exists. Second, to evaluate RPi resource consumption during each test, in terms of total CPU consumption with memory usage as an additional metric.

Initially, Metricbeat was used to collect these metrics but due to service disconnect and correct time synchronization issues, a local script was written to collect these metrics in real time. Instead replaying already captured traffic from pcaps using Tcpreplay, actual attacks using Kali Linux were preformed in every test. The reason of choosing real attacks was to sketch a more realistic attack scenario and accumulate more accurate data.

Test 1 used syn-flood at 100MBit/sec data rate with smaller packets, purpose of the attack was to put Snort on work into detection of this attack as well as consume most of the bandwidth of Ethernet interface eth0, as a result increase the CPU usage of RPi. Test increased the CPU to an average of 56% which had no mention-able effects on RPi performance but Snort gave a moderate performance of analyzing only 7.42%. Test 2 and Test 3 gave almost similar Snort statistics except test 3, which clocked the RPi CPU to 96%.

Test 4 was a worst case test scenario where both of the interfaces were attacked simultaneously. This test clocked the CPU to it's full potential with an average of 98.825%. Nonetheless, during these tests, CPU of RPi was tested rigorously, which showed an average performance, due to the fact that even at high CPU rate no system crash, Rpi shutdown or a major system disruption was noted and Rpi was still able to perform it's system operations. Similarly, Snort gave below average performance in detection of high rate attacks, and excellent performance

in detection of low rate attacks i.e. detection of wlan0 attack with 99.75% analysis rate. These tests signify that Rpi can perform as an IDS using Snort in average traffic flow environments. This also provides the answer to the problem statement question i.e *Does RPi have enough resources to support Snort as an IDS, in terms of resource consumption?* and the answer is yes.

These tests uncovered a major weakness of RPi which is the Ethernet interface. Rpi's Ethernet provides a maximum bandwidth support of only 300Mbps which is very low for modern internet traffic scenarios. Although it's Ethernet is designed for sensor networks where less data flow is expected but RPi does have the capability to operate as an IDS in non-sensor based networks, for that, it will perhaps, need a better Ethernet. Except test 1, every other test completely engaged Ethernet port and no incoming or outgoing communication was possible. Beats were the biggest victims of Ethernet's inability of communication during these tests. This may gave an idea of using alternative single board computers with a better Ethernet port support which may cost relatively higher price. For instance, *Orange Pi Plus 2* and *Asus Tinker Board* provides one Gigabit Ethernet port with only a marginal price increase. It is recommended to avoid using the RPi in cases where a high data rate traffic is expected. If, in case it is unavoidable then RPi alternatives with better specifications could be used.

In [46], authors presented a similar approach using RPi model 2 B. They came to almost similar conclusion that Ethernet interface of RPi can be a major weakness, but authors did not provide a worst case scenario neither they could provide a clear view of their system design. In [39] authors used behaviour profiling, which is extremely difficult to implement in common IoT setup, similarly in [40], authors used artificial immune cells which can be computational and resource overhead while [41] is limited to detect only few special attacks. In [42], authors used only a single metric for detecting attacks i.e. energy consumption. Also [43] was designed for only 6LoWPAN while [44] proposed a hybrid approach which is too complex to implement in common IoT environments. Finally, except [46] none of the authors could provide cost affordability of their proposals.

This project not only identified best to worst case scenarios but also provided a clear system design with cost affordability. Moreover, this project has provided probably the first practical implementation of one of the most advanced yet free monitoring service ELk stack, on Rpi with open-source IDS Snort.

Short coming of this project probably is the inability of detecting direct wireless attacks on wlan0 access point. A de-authentication attack on RPi access point will disconnect any connected IoT devices and the attack will go unnoticed of Snort instance. Since this attack is very rare and need a very close range to initiate the attack, it can be considered less probable.

6.5 Future Work

The project can be improved by changing several parameters. For instance, instead using two instances of Snort, wlan0 instance of Snort can be changed with Kismet IDS instead which is a specialized wireless IDS. This setup will then provide intrusion detection on Ethernet interface with Snort while Kismet will take care of wlan0 interface.

For a pure Wireless IDS in a wider IoT environment, multiple battery powered RPi's with Kismet can be deployed with a distributed approach i.e. in a cluster, where each device cooperates with it's neighbour devices, by exchanging information of attack detection, battery levels, and load balancing etc. These cluster of IDS devices can be programmed to autonomously choose their cluster heads for instance those devices with higher battery can be chosen as cluster head or some other parameter can be set for this. For monitoring of battery consumption and power levels, Beats can be deployed with a longer data shipping interval to save power.

It is also possible to make these clustered IDS sensors autonomous in decision making and load distribution by introducing very basic machine learning using python. Moreover, this is also possible to turn Snort IDS into an IPS (intrusion prevention system) but it may not work good in wireless environments. Other open-source IDSs like Bro and Suricata can also be used as an alternative to Snort.

Chapter 7

Conclusion

This thesis proposed an IDS system for IoTs using very common, low price and low spec hardware with freely available open source intrusion detection system. The purpose of the thesis was to open new paths into IoT security by practically demonstrating that security of IoTs in terms of Intrusion detection can be achieved by using simple and affordable tools. It provided an affordable alternative to costly commercial IDS systems which are not feasible for common IoT environments like smart home and smart buildings.

The proposed IDS can be deployed in most of the IoT environments where data traffic is expected to be low to medium, for instance, in smart buildings, public transport stations or in personal home security. To prove the claims of the project, it was evaluated and tested under various data traffic rates and by using different types of real attacks. Best to worst case scenarios were implemented to test RPi's capability of hosting Snort, tests result confirmed the feasibility of Raspberry Pi as an IDS and proved that under most common IoT environments, Raspberry Pi can host Snort and can deliver the expected intrusion detection functionality.

Although certain tests increased the RPi CPU to almost 100% but it was not entirely due to Snort processes but was associated with data rates. Raspberry Pi hosted Snort smoothly with no significant impact on memory neither CPU shutdowns were noted during any of the tests, although It would be a good idea to further test the proposed system. This project has successfully built an affordable intrusion detection system for IoTs and has managed to test and prove it's claims. Also, this was the first project which used a combination of Snort, Raspberry Pi and ELK Stack.

Bibliography

[1] K. Ashton. That 'internet of things' thing. RFiD Journal, 22:97–114, 2009.

[2] https://www.forbes.com/sites/louiscolumbus/2017/12/10/2017-roundup-of-internet-of-things-forecasts/#3fc008701480 [Accessed September 3 2018]

[3] Introduction: Advances in IoT research and applications, Pan Wang & Ricardo Valerdi & Shangming Zhou & Ling Li, Published online: 11 March 2015 Springer Science+Business Media New York 2015.

[4] https://www.cisco.com/c/dam/en_us/about/ac79/docs/retail/Beyond-the-New-Normal_IBSG_051211-FINAL.pdf [Accessed September 3 2018]

[5] http://sro.sussex.ac.uk/54129/1/07110295.pdf [Accessed September 5 2018]

[6] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler, "Transmission of IPv6 packets over IEEE 802.15.4 networks," in Internet Engineering Task Force, RFC 4944, 2007.

[7] T. W. (Ed.), P. T. (Ed.), A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister,R. Struik, J. Vasseur, and R. Alexander, "RPL: IPv6 routing protocol for low-power and lossy networks," in IETF, RFC 6550, 2012.

[8] Z. Shelby, K. Hartke, and C. Bormann, "Constrained application protocol (CoAP)," Internect Draft, Available at: http://datatracker.ietf.org/wg/core/charter.

[9] Z. Sheng, S. Yang, Y. Yu, A. V. Vasilakos, J. A. McCann, and K. K.Leung, "A survey on the IETF protocol suite for the internet of things: standards, challenges, and opportunities," IEEE Wireless Communications Magazine, vol. 20, no. 6, pp. 91–98, 2013.

[10] European Telecommunication Standards Institute, (ETSI),http://www.etsi.org/technologies-clusters/technologies/m2m [Accessed September 5 2018]

[11] D. Katusic, M. Weber, I. Bojic, G. Jezic, and M. Kusek, “Market, standardization, and regulation development in machine-to-machine communications,”in 20th International Conference on Software, Telecommunications and Computer Networks (SoftCOM), Sept 2012, pp. 1–7.

[12] Internet of Things: Innovation with Chinese Characteristics, http://www.hoganlovells.com/internet-of-things-innovation-withchinese-characteristics-09-12-2013/ [Accessed September 6 2018]

[13] https://iot-analytics.com/top-10-iot-segments-2018-real-iot-projects/ [Accessed September 6 2018]

[14] http://sro.sussex.ac.uk/54129/1/07110295.pdf [Accessed September 6 2018]

[15] https://altizon.com/varroc-automotive-case-study/ [Accessed September 10 2018]

[16] OASIS, MQTT Version 3.1.1, Std., 2014.

[17] Dae-Hyeok Mun, Minh Le Dinh, and Young-Woo Kwon, Department of Computer Science, "An Assessment of Internet of Things Protocols for Resource-Constrained Applications," IEEE 40th Annual Computer Software and Applications Conference 2016.

[18] Z. Shelby, K. Hartke, and C. Bormann, “The constrained application protocol (CoAP),” 2014.

[19] I. Fette and A. Melnikov, “The Websocket Protocol,” 2011.

[20] https://xmpp.org/about/ [Accessed September 12 2018]

[21] Q. Jing, A. V. Vasilakos, J. Wan, J. Lu, and D. Qiu, “Security of the Internet of Things: Perspectives and challenges,” Wireless Netw., vol. 20, no. 8, pp. 2481–2501, Nov. 2014.

[22] M. Frustaci, P. Pace, G. Aloi and G. Fortino, "Evaluating Critical Security Issues of the IoT World: Present and Future Challenges," in IEEE Internet of Things Journal, vol. 5, no. 4, pp. 2483-2495, Aug. 2018.

[23] https://www.owasp.org/index.php/OWASP_Internet_of_Things _Project#tab=IoT_Attack_Surface_Areas. [Accessed September 12 2018]

[24] C. Karlof, D. Wagner, “Secure Routing in Wireless Sensor Networks: Attacks and Countermeasures”, 2003

[25] https://www.sans.org/reading-room/whitepapers/detection/understanding-intrusion-detection-systems-337 [Accessed September 12 2018]

[26] Steven R Snapp et al.“DIDS (distributed intrusion detection system) motivation, architecture, and an early prototype.” In: Proceedings of the 14th

national computer security conference. Vol. 1. Washington, DC. 1991, pp. 167–176.

[27] https://snort.org/ [Accessed September 15 2018]

[28] https://www.bro.org/sphinx/intro/index.html [Accessed September 26 2018]

[29] https://redmine.openinfosecfoundation.org/projects/suricata/wiki /What_is_Suricata [Accessed September 26 2018]

[30] F. Hock and P. Kortiš, "Commercial and open-source based Intrusion Detection System and Intrusion Prevention System (IDS/IPS) design for an IP networks," 2015 13th International Conference on Emerging eLearning Technologies and Applications (ICETA), Stary Smokovec, 2015, pp. 1-4.

[31] A. Mishra, K. Nadkarni, and A. Patcha, "Intrusion detection in wireless ad hoc networks", IEEE Wireless Communications, 11 (1), 48-60, 2004.

[32] T. Anantvalee, and W. Jie, "A survey on intrusion detection systems in mobile ad hoc networks", Wireless Network Security, 2, 159-180, 2017.

[33] S. Kumar, and K. Dutta, "Intrusion detection in mobile ad hoc networks: techniques, systems, and future challenges", Security and Communication Networks, 9 (14), 2484-2556, 2016

[34] A. Farooqi, and F. Khan, "Intrusion detection systems for wireless sensor networks: a survey", In Communication and Networking Communications in Computer and Information Science, 56, Springer, Berlin, Heidelberg, 234-241, 2009

[35] A. Abduvaliyev, A. Pathan, Z. Jianying, R. Roman, and W. WaiChoong2013, "On the vital areas of intrusion detection systems in wireless sensor networks", IEEE Communications Surveys & Tutorials, 15 (3), 1223-1237, 2013.

[36] I. Butun, S. Morgera, and R. Sankar, "A survey of intrusion detection systems in wireless sensor networks", Communications Surveys and Tutorials IEEE, 16 (1), 266-282, 2014.

[37] R. Mitchell, and I. Chen, "A survey of intrusion detection techniques for cyber-physical systems", ACM Computing Surveys (CSUR), 46 (4), 55, 2014.

[38] B. B. Zarpelão, R. S. Miani, C. T. Kawakani, and S. C. de Alvarenga, "A survey of intrusion detection in internet of things," Journal of Network and Computer Applications, vol. 84, pp. 25-37, 2017.

[39] E. Cho, J. Kim, and C. Hong, "Attack model and detection scheme for botnet on 6LoWPAN," In Management Enabling the Future Internet for Changing Business and New Computing Services, Lecture Notes in Computer Science 5787. Springer, Berlin, Heidelberg, 515-518, 2009.

[40] C. Liu, J. Yang, Y. Zhang, R. Chen, and J. Zeng, "Research on immunity-based intrusion detection technology for the Internet of Things," In: Natural Computation (ICNC), 2011 Proceedings of the Seventh International Conference, Vol. 1, pp. 212-216, 2011.

[41] S. Raza, L. Wallgren, and T. Voigt, "SVELTE: real-time intrusion detection in the Internet of Things," Ad Hoc Network, 11 (8), 2661-2674, 2013.

[42] T. Lee, C. Wen, L. Chang, H. Chiang, and M. Hsieh, "A lightweight intrusion detection scheme based on energy consumption analysis in 6LowPAN," In: Advanced Technologies, Embedded and Multimedia for Human-centric Computing, Lecture Notes in Electrical Engineering, 260. Springer, Netherlands, 1205-1213, 2014.

[43] C. Cervantes, D. Poplade, M. Nogueira, and A. Santos, "Detection of sinkhole attacks for supporting secure routing on 6LoWPAN for Internet of Things," In: 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM), pp. 606-611, 2015

[44] A. Le, J. Loo, K. Chai, and M. Aiash, "A specification-based IDS for detecting attacks on RPL-based network topology," Information, 7 (2), 25, 2016.

[45] D. Midi, A. Rullo, A. Mudgerikar, and E. Bertino, "Kalis: A system for knowledge-driven adaptable intrusion detection for the Internet of Things," In Proceedings of the IEEE 37th International Conference on Distributed Computing Systems (ICDCS'17), 2017

[46] A. Sforzin, F. G. Mármol, M. Conti and J. Bohli, "RPiDS: Raspberry Pi IDS — A Fruitful Intrusion Detection System for IoT," 2016 Intl IEEE, Toulouse, 2016, pp. 440-448.

[47] https://www.zigbee.org/ [Accessed October 5 2018]

[48] https://wave2m.com.cutestat.com/ [Accessed October 5 2018]

[49] https://www.isa.org/standards-publications/isa-publications/intech-magazine/2012/december/web-exclusive-analysis-wireless-industrial-automation-standards-isa-100-11a-wirelesshart/ [Accessed October 5 2018]

[50] https://www.profibus.com/ [Accessed October 5 2018]

[51] https://www.omg.org/ [Accessed October 5 2018]

[52] http://malware.wikia.com/wiki/Sub7 [Accessed October 10 2018]

[53] https://hackstory.net/BackOrifice [Accessed October 10 2018]

[54] https://nmap.org/ [Accessed October 10 2018]

[55] https://sectools.org/tool/l0phtcrack/ [Accessed October 10 2018]

[56] https://github.com/Snorby/snorby [Accessed October 10 2018]

[57] http://acidlab.sourceforge.net/ [Accessed October 10 2018]

[58] https://www.elastic.co/elk-stack [Accessed October 10 2018]

[59] https://sourceforge.net/projects/secureideas/ [Accessed October 10 2018]

[60] http://www.tcpdump.org/manpages/tcpdump.1.html [Accessed October 10 2018]

[61] https://www.mcafee.com/enterprise/en-us/products/network-security-platform.html [Accessed October 18 2018]

[62] https://www.trendmicro.com/en_us/business/products/network/intrusion-prevention/tipping-point-threat-protection-system.html [Accessed October 18 2018]

[63] https://www.hillstonenet.com/products/network-intrusion-prevention-system-s-series/ [Accessed October 18 2018]

[64] https://www.gartner.com/reviews/market/intrusion-prevention-systems/vendor/huawei/product/network-intelligent-protection-nip-system. [Accessed October 18 2018]

[65] https://www.raspberrypi.org/ [Accessed October 22 2018]

[66] Alfred V. Aho and Margaret J. Corasik. Efficient String Matching: An Aid to Bibliographic. Search: Bell Labs, Communications of the ACM Jun 1975 Volume 18 Number 6.

[67] https://snort-org-site.s3.amazonaws.com/production/document_files/files/000/000/085/original/OptimizingPathernMatchingForIDS.pdf [Accessed October 25 2018]

[68] https://growthenabler.com/flipbook/pdf/IOT%20Report.pdf [Accessed September 25 2018]

[69] https://rules.emergingthreats.net/open/snort-2.9.0/rules/ [Accessed October 27 2018]

[70] https://www.snort.org/talos [Accessed October 27 2018]

[71] https://www.snort.org/downloads#rules [Accessed October 27 2018]

[72] https://www.kismetwireless.net/ [Accessed October 29 2018]

[73] https://golang.org/dl/ [Accessed October 29 2018]

[74] https://www.kali.org [Accessed October 29 2018]

[75] https://www.elastic.co/elk-stack [Accessed November 6 2018]

[76] http://www.orangepi.org/orangepiplus2/ [Accessed November 6 2018]

[77] https://www.asus.com/us/Single-Board-Computer/Tinker-Board/ [Accessed October 4 2018]

[78] http://www.banana-pi.org/m3.html [Accessed October 4 2018]

[79] http://en.t-firefly.com/product/rk3288 [Accessed October 4 2018]

[80] https://www.fs-net.de/en/products/armstone/armstonea9/ [Accessed October 4 2018]

[81] https://www.kismetwireless.net/ [Accessed November 8 2018]

Appendices

Appendix A

Install and Configure Snort

```
#update repositories
sudo apt-get update && sudo apt-get dist-upgrade -y
# install Snort dependencies
sudo apt-get install build-essential -y
sudo apt-get install libnet1-dev -y
sudo apt-get install checkinstall
sudo apt-get install bison git -y
sudo apt-get install flex -y
sudo apt-get install libpcap-dev -y
sudo apt-get install libpcre3-dev -y
sudo apt-get install libdumbnet-dev -y
# Create a directory for Snort
sudo mkdir snort_src && cd snort_src
# Download the source Snort packages and DAQ libraries.
sudo wget https://www.snort.org/downloads/snort/snort-2.9.12.tar.gz
# Current version by date
sudo wget https://www.snort.org/downloads/snort/daq-2.0.6.tar.gz
sudo tar xvzf daq-2.0.6.tar.gz && cd daq-2.0.6/
sudo ./configure
sudo make
# provide package description as snort-daq and keep all the other options
    as default.
sudo checkinstall -D --install=no --fstrans=no
# A package is created in same directory with .dab extension, run it as..
sudo dpkg -i daq_2.0.6-1_armhf.deb
cd ..
sudo tar xvfz snort-2.9.12.tar.gz && cd snort-2.9.12/
sudo ./configure --enable-sourcefire
sudo make
sudo checkinstall -D --install=no --fstrans=no
# follow same process as daq, another .deb package is created for snort,
```

```
    run it as:
sudo dpkg -i snort_2.9.12-1_armhf.deb
sudo ldconfig
# confirm Snort installation directory by:
which snort
# create a symlink to snort directory
sudo ln -s /usr/local/bin/snort /usr/sbin/snort
#check snort installation
snort --version
# Add snort group and user
sudo groupadd snort
sudo useradd snort -r -s /sbin/nologin -c SNORT_IDS -g snort
# create snort work directories
sudo mkdir -p /etc/snort/rules/iplists
sudo mkdir /etc/snort/preproc_rules
sudo mkdir /usr/local/lib/snort_dynamicrules
sudo mkdir /etc/snort/so_rules
sudo touch /etc/snort/rules/local.rules
sudo touch /etc/snort/rules/iplists/white_list.rules
sudo touch /etc/snort/rules/iplists/black_list.rules
sudo mkdir -p /var/log/snort/eth0
sudo mkdir /var/log/snort/wlan0
# provide permission to snort user to access directories
sudo chmod -R 5775 /etc/snort
sudo chmod -R 5775 /var/log/snort
sudo chmod -R 5775 /usr/local/lib/snort_dynamicrules
sudo chown -R snort:snort /etc/snort
sudo chown -R snort:snort /var/log/snort
sudo chown -R snort:snort /usr/local/lib/snort_dynamicrules
# move snort config packages to working directory /etc
cd snort_src/snort-2.9.12/etc
sudo cp *.conf /etc/snort
sudo cp *.config /etc/snort
sudo cp *.map /etc/snort
#further snort.conf configurations will be detailed in upcoming sections
#Test run snort
sudo snort -T -c /etc/snort/snort.conf -i eth0
```

Appendix B

Hostapd and ISC server configurations

```
sudo apt-get update
# install required packages
sudo apt-get install hostapd isc-dhcp-server
sudo apt-get install iptables-persistent
# Edit dhcpd.conf
sudo nano /etc/dhcp/dhcpd.conf
# comment of these lines with "#"
#option domain-name "example.org";
#option domain-name-servers ns1.example.org, ns2.example.org;
# uncomment #authoritative;
authoritative;
# Add following at the end of file, change ip's accordingly
subnet 192.168.42.0 netmask 255.255.255.0 {
        range 192.168.42.1 192.168.42.40;
        option broadcast-address 192.168.42.255;
        option routers 192.168.42.1,192.168.10.1;
        default-lease-time 600;
        max-lease-time 7200;
        option domain-name "local";
        option domain-name-servers 8.8.8.8,
            8.8.4.4,192.168.42.1,192.168.10.1;
}
# Edit isc-dhcp-server file and add
sudo nano /etc/default/isc-dhcp-server
INTERFACESv4="wlan0"
# make sure wlan0 is down
sudo ifdown wlan0
# Edit file /etc/network/interfaces.d/wlan0
sudo nano /etc/network/interfaces.d/wlan0
```

```
# Add following and save file
allow-hotplug wlan0
iface wlan0 inet static
  address 192.168.42.1
  netmask 255.255.255.0
# Time to enable configure hostapd, hostapd.conf does not exist by
    default
sudo nano /etc/hostapd/hostapd.conf
# Add following code
interface=wlan0
ssid=Pi_AP   # any AP name
country_code=NO
hw_mode=g
channel=6
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
wpa=2
wpa_passphrase=yourpassword
wpa_key_mgmt=WPA-PSK
wpa_pairwise=CCMP
wpa_group_rekey=86400
ieee80211n=1
wme_enabled=1
# Edit /etc/default/hostapd
sudo nano /etc/default/hostapd
# and add following
DAEMON_CONF="/etc/hostapd/hostapd.conf"
# Enable ipv4 forwarding
sudo nano /etc/sysctl.conf
# and add at the end of file
net.ipv4.ip_forward=1
# Run command
sudo sh -c "echo 1 > /proc/sys/net/ipv4/ip_forward"
# Add iptables rules
sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
sudo iptables -A FORWARD -i eth0 -o wlan0 -m state --state RELATED,
    ESTABLISHED -j ACCEPT
sudo iptables -A FORWARD -i wlan0 -o eth0 -j ACCEPT
# verify iptable rules by
sudo iptables -t nat -S
sudo iptables -S
# save iptable rules persistently
sudo sh -c "iptables-save > /etc/iptables/rules.v4"
# turn on wlan0 and restart
sudo ifup wlan0
sudo reboot
# start services
sudo service hostapd start
sudo service isc-dhcp-server start
```

```
# verify status
sudo service hostapd status
sudo service isc-dhcp-server status
# Follow commands can also be used to start daemon services
sudo update-rc.d hostapd enable
sudo update-rc.d isc-dhcp-server enable
```

Appendix C

Configuring Beats on RPi

```
# Install dependencies
sudo apt-get install python-pip git
sudo pip install virtualenv
sudo wget https://dl.google.com/go/go1.11.2.linux-armv6l.tar.gz
sudo tar -C /usr/local/ -xzf go1.11.2.linux-armv6l.tar.gz
export PATH=$PATH:/usr/local/go/bin
sudo nano ~/.profile
# add following in file,if does not work, try adding same to /etc/bash.
    bashrc file also.
#! /bin/bash
export PATH=$PATH:/usr/local/go/bin # and close the file
sudo chmod a+x ~/.profile # or /etc/bash.bashrc
PS1 '$ ' source ~/.profile
# verify Go
go version
# if "go command not found" error is thrown, provide the path again
export PATH=$PATH:/usr/local/go/bin
# setting up beats
sudo mkdir go
export GOPATH=$HOME/go
cd go
sudo mkdir -p src/github.com/elastic
cd src/github.com/elastic
sudo git clone https://github.com/elastic/beats.git
cd beats
# before installing any beats component, better to increase swap memory
    for RPi, otherwise "low memory" error will be thrown.
sudo nano /etc/dphys-swapfile
# change following

CONF_SWAPFILE=/var/swap
```

```
CONF_SWAPSIZE=2048
CONF_MAXSWAP=3000 # save and exit
swapon /var/swap  # or /etc/init.d/dphys-swapfile restart
# verify swap by
swapon -s
# provide permissions
sudo chown -R pi $HOME/go  # incase of user pi
sudo chown -R pi /usr/local/go
# For configuring filebeat
cd filebeat
make                        # Do not use sudo
# Metricbeat can also be installed by following same process
# Edit filebeat.yml and change/enable following:
sudo nano filebeat.yml
filebeat.inputs:
- type: log
  enabled: true
  paths:
    - /var/log/snort/eth0/alert
    - /var/log/snort/wlan0/alert
  event.type: snort
setup.kibana:
    host: "ip address:5601"

# disable elasticsearch output and enable logstash
output.logstash:
    hosts: ["ip address:5044"] # save and exit
```

Appendix D

Setup ELK on Openstack

```
#The Server should have at leas 2 CPU cores and 2GB RAM
# Installing Java
sudo apt install software-properties-common apt-transport-https -y
sudo add-apt-repository ppa:webupd8team/java -y
sudo apt install oracle-java8-installer -y
# verify Java installation
java -version
# Setup Java environment
update-alternatives --config java
# Create the profile file under profile.d directory
sudo nano /etc/profile.d/java.sh
# Add following
#Set JAVA_HOME
JAVA_HOME="/usr/lib/jvm/java-8-oracle"
export JAVA_HOME
PATH=$PATH:$JAVA_HOME
export PATH
# Provide permissions and source the file
chmod +x /etc/profile.d/java.sh
source /etc/profile.d/java.sh
# Verify Java environment
echo $JAVA_HOME
# Installing Elasticsearch
wget -qO - https://artifacts.elastic.co/GPG-KEY-elasticsearch | sudo apt-
    key add -
echo "deb https://artifacts.elastic.co/packages/6.x/apt stable main" |
    sudo tee -a /etc/apt/sources.list.d/elastic-6.x.list
sudo apt update
sudo apt install elasticsearch -y
# Configure elasticsearch.yml file
cd /etc/elasticsearch/
```

```
sudo nano elasticsearch.yml
# For non-root users, elasticsearch.yml can not be accessed. in such case, use:
sudo chown -R ubuntu /etc/elasticsearch # ubuntu is the current user, in this case
# change following in elasticsearch.yml
network.host: localhost
http.port: 9200           # save file and exit.
# start elasticsearch service and verify
systemctl start elasticsearch
systemctl enable elasticsearch
netstat -plntu
curl -XGET 'localhost:9200/?pretty'
# Install and configure Kibana
sudo apt install kibana -y
cd /etc/kibana/
sudo nano kibana.yml
server.port: 5601
server.host: "localhost"
elasticsearch.url: "http://localhost:9200" # save and exit
# start Kibana service and verify port
sudo systemctl enable kibana
sudo systemctl start kibana
netstat -plntu

# Since Kibana is configured to listen on localhost, a reverse proxy is needed to allow remote access.Nginx is used for this purpose.
apt-get -y install nginx apache2-utils -y
# upadte nginx virtual host file, /etc/nginx/sites-available/default, change SERVERIP by your IP
server {
        listen     80;
        server_name SERVERIP;
        return 301 https://;
}

server {
        listen              *:443 ;
        ssl on;
        ssl_certificate /etc/pki/tls/certs/logstash-forwarder.crt;
        ssl_certificate_key /etc/pki/tls/private/logstash-forwarder.key;
        server_name         SERVERIP;
        access_log          /var/log/nginx/kibana.access.log;
        error_log /var/log/nginx/kibana.error.log;

location / {
        auth_basic "Restricted";
        auth_basic_user_file /etc/nginx/htpasswd.users;
        proxy_pass http://localhost:5601;
}
```

```
}
# save and exit the file
# Generating password for kibana
sudo htpasswd -c /etc/nginx/.kibana-user KibanaUser # provide the
    password and also change KibanaUser
ln -s /etc/nginx/sites-available/kibana /etc/nginx/sites-enabled/ nginx -
    t
# Enable nginx
systemctl enable nginx
systemctl restart nginx
#Install and configure Logstash
apt-get update
apt-get -y install logstash

#Generate SSL Certificates, since Filebeat is used to ship logs from RPi
    to ELK Server, it is need to create an SSL certificate and key pair.
    The certificate is used by Filebeat to verify the identity of ELK
    Server. It is optional.
mkdir -p /etc/pki/tls/certs
mkdir /etc/pki/tls/private

sed -i "230i subjectAltName = IP: ${server_address}" /etc/ssl/openssl.cnf
cd /etc/pki/tls
openssl req -config /etc/ssl/openssl.cnf -x509 -days 3650 -batch -nodes -
    newkey rsa:2048 -keyout private/logstash-forwarder.key -out certs/
    logstash-forwarder.crt
cd

# Configure Logstash files, i.e. input, output and filter.
# configuring /etc/logstash/conf.d/02-beats-input.conf
sudo nano /etc/logstash/conf.d/02-beats-input.conf
input {
  beats {
    port => 5044
#    ssl => true
#    ssl_certificate => "/etc/pki/tls/certs/logstash-forwarder.crt"
#    ssl_key => "/etc/pki/tls/private/logstash-forwarder.key"
    type => "Snort"
  }
}
# save and exit
# configure Snort filter file
sudo nano 11-snort-filter.conf
filter {
  if [type] == "Snort" {
    grok { match => { "message" => "\[\*\*\] \[%{GREEDYDATA:Signature}\]
        %{GREEDYDATA:SnortMessage} \[\*\*\]%{GREEDYDATA:Classification}\[
        Priority: %{NUMBER:priority:int}\] \n%{GREEDYDATA:snortDate} %{IP
        :SourceIP}:%{NUMBER:SourcePort:int} \-\> %{IP:DestinationIP}:%{
        NUMBER:DestinationPort:int}\n%{WORD:Protocol} TTL:%{NUMBER:TTL:
```

```
        int} TOS:%{BASE16NUM:TOS} ID:%{NUMBER:ID:int} IpLen:%{NUMBER:
        IpLen:int} %{GREEDYDATA:options}"}
#     remove_field => [ "message" ]
    }

    date {
      match => ["snortDate", "MM/dd-HH:mm:ss.SSSSSS"]
      target => "@timestamp"
      remove_field => ["snortDate"]
      timezone => "Europe/Amsterdam"
      locale => "en_US"
    }
    geoip{
      source=>"SourceIP"
    }
  }
}
# Configure elasticsearch output file
sudo nano 30-elasticsearch-output.conf
output {
  if [type] == "Snort" {
    elasticsearch {
      hosts => ["localhost:9200"]
      manage_template => false
      index => "snort-%{+YYYY.MM.dd}"
    }
  }
}
# save and exit

# Verify and reboot setup
java -version
service nginx restart
service elasticsearch restart
service kibana restart
service logstash restart
systemctl enable elasticsearch
systemctl enable logstash
systemctl enable kibana
```

Appendix E

Snort Instances and Gnuplot

```
1
2  # snort.sh file, Run by bash snort.sh
3  # No need to run stats.sh file separately for plotting
4
5  #! bin/bash
6  #Script to initiate Snort instances from given options.
7  #It will open new terminal (tested only in RPi Raspbian stretch) and run
       the given instances.
8  # Author: HH@Khan
9
10 RED='\033[0;31m'
11 YEL='\033[0;93m'
12 GRE='\033[0;32m'
13 NOCO='\033[0m'
14
15 while true; do
16        echo -ne "${GRE}Choose option to run snort and beats--------: \n\
              t${YEL}[1]: ${NOCO}Run Snort on Eth0\n\t${YEL}[2]: ${NOCO}Run
               Snort on wlan0\n\t${YEL}[3]: ${NOCO}Run Snort on both eth0
              and wlan0 \n\t${YEL}[4]: ${NOCO}Run Filebeat and Metricbeat\n
              \t${YEL}[5]: ${NOCO}Run Snort on eth0 with beats\n\t${YEL
              }[6]: ${NOCO}Run both snort instances and beats\n\t"
17
18        read OPTION
19   case $OPTION in
20       1 )
21        echo -ne "\n\t\t${GRE}Running snort on eth0
              ------------------------------------"
22        lxterminal --command="/bin/bash -c 'bash stats.sh; /bin/bash'" #
              Calling Plotting file before instance run.
23        sleep 5                          # plotting file starts 5 seconds
```

```
            before Snort instances
24          lxterminal --command="/bin/bash -c 'sudo snort -c /etc/snort/
                snort_eth0.conf -l /var/log/snort/eth0;/bin/bash'"
25                  lxterminal --command="/bin/bash -c 'sudo tail -f /var
                        /log/snort/eth0/alert;/bin/bash'"
26
27          sleep 600      # Run Time of Snort
28          sudo pkill -SIGINT snort
29          break
30          ;;
31        2 )
32          echo -ne "\n\t${GRE}Running Snort on wlan0
                ------------------------------------"
33          sleep 5
34          lxterminal --command="/bin/bash -c 'sudo snort -c /etc/snort/
                snort_wlan0.conf -l /var/log/snort/wlan0 -i wlan0;/bin/bash'"
35          lxterminal --command="/bin/bash -c 'sudo tail -f /var/log/snort/
                wlan0/alert;/bin/bash'"
36          lxterminal --command="/bin/bash -c 'bash stats.sh; /bin/bash'"
37          sleep 600
38          sudo pkill -SIGINT snort
39          break
40          ;;
41        3 )
42          echo -ne "\n\t${GRE}Runnig Snort on both eth0 and wlano
                ------------------------"
43          lxterminal --command="/bin/bash -c 'bash stats.sh; /bin/bash'"
44          sleep 5
45          lxterminal --command="/bin/bash -c 'sudo snort -c /etc/snort/
                snort_wlan0.conf -l /var/log/snort/wlan0 -i wlan0;/bin/bash'"
46          lxterminal --command="/bin/bash -c 'sudo snort -c /etc/snort/
                snort_eth0.conf -l /var/log/snort/eth0 -i eth0;/bin/bash'"
47
48          lxterminal --command="/bin/bash -c 'sudo tail -f /var/log/snort/
                eth0/alert;/bin/bash'"
49          lxterminal --command="/bin/bash -c 'sudo tail -f /var/log/snort/
                wlan0/alert;/bin/bash'"
50
51          sleep 600
52          sudo pkill -SIGINT snort
53          sleep 1
54          sudo pkill -SIGINT snort
55          break
56          ;;
57        4 )
58          echo -ne "\n\t${GRE}Running Filebeat and Metricbeat...\n"
59          sleep 5
60          lxterminal --command="/bin/bash -c 'cd /home/pi/go/src/github.com
                /elastic/beats/filebeat && ./filebeat.sh -e;/bin/bash'"
61            lxterminal --command="/bin/bash -c 'cd /home/pi/go/src/github.
```

```
          com/elastic/beats/metricbeat && ./metricbeat -e;/bin/bash'"
62        break
63        ;;
64      5 )
65
66        echo -ne "\n\t${GRE}Running snort on eth0 and beats--------"
67        lxterminal --command="/bin/bash -c 'bash stats.sh; /bin/bash'"
68        sleep 5
69        lxterminal --command="/bin/bash -c 'sudo snort -c /etc/snort/
              snort_eth0.conf -l /var/log/snort/eth0;/bin/bash'"
70           lxterminal --command="/bin/bash -c 'sudo tail -f /var/log/
                 snort/eth0/alert;/bin/bash'"
71        lxterminal --command="/bin/bash -c 'cd /home/pi/go/src/github.com
              /elastic/beats/filebeat && ./filebeat.sh -e;/bin/bash'"
72           lxterminal --command="/bin/bash -c 'cd /home/pi/go/src/
                 github.com/elastic/beats/metricbeat && ./metricbeat -e
                 ;/bin/bash'"
73
74        sleep 600
75           sudo pkill -SIGINT snort
76        break
77        ;;
78       6)    echo -ne "\n\t${GRE}Running both snort instances and beats
             --------"
79        lxterminal --command="/bin/bash -c 'bash stats.sh; /bin/bash'"
80        sleep 4
81           lxterminal --command="/bin/bash -c 'sudo snort -c /etc/
                 snort/snort_eth0.conf -l /var/log/snort/eth0; /bin/
                 bash'"
82           lxterminal --command="/bin/bash -c 'sudo tail -f /var/log/
                 snort/eth0/alert; /bin/bash'"
83        lxterminal --command="/bin/bash -c 'sudo snort -c /etc/snort/
              snort_wlan0.conf -l /var/log/snort/wlan0 -i wlan0; /bin/bash'
              "
84           lxterminal --command="/bin/bash -c 'sudo tail -f /var/log/
                 snort/wlan0/alert; /bin/bash'"
85           lxterminal --command="/bin/bash -c 'cd /home/pi/go/src/
                 github.com/elastic/beats/filebeat && ./filebeat.sh -e
                 ;/bin/bash'"
86           lxterminal --command="/bin/bash -c 'cd /home/pi/go/src/
                 github.com/elastic/beats/metricbeat && ./metricbeat -e
                 ;/bin/bash'"
87        sleep 600
88        sudo pkill -SIGINT snort
89        sleep 1
90           sudo pkill -SIGINT snort
91           break
92        ;;
93
94      * )
```

```
95          echo -ne "\n\t${RED}[-] ERROR:${NOCO} Invalid option\n\n"
96          ;;
97      esac
98
99  done
100 ##################################################
101 #content of file stats.sh which is called from within snort.sh script
102 #! /bin/bash
103 # Script to get CPU and Memory every next second
104 #It will also plot the results after provided time using Gnuplot
105 # Author HH@Khan
106
107 sudo rm status.dat
108
109 function mem_cpu() {
110
111 end=$((SECONDS+605))
112
113 echo "Printing memory and cpu consumption in status.dat file..... "
114 while [ $SECONDS -lt $end ]; do
115
116 MEMORY=$(free -m | awk 'NR==2{printf "%.2f\t\t", $3*100/$2 }')
117 CPU=$(echo $[100-$(vmstat 1 2|tail -1|awk '{printf $15}')])
118
119 echo -e "$SECONDS\t\t$MEMORY$CPU" >>status.dat
120 sleep 1
121 done
122 }
123
124 function gnu_plot() {
125 sudo gnuplot -persist <<-EOFMarker
126 set y2tics
127 # We don't want to see the left ticks on the right axis
128 set ytics nomirror
129
130 # Set ranges so that the data points are not on the axis
131 set xrange [0:620] # "set autoscale x" can also be used. It presents Time
        in Seconds
132
133 set yrange [0:110] #Percentages
134 set y2range[0:110]
135
136 # use first line of the file for labels
137 set key autotitle columnhead
138 # display key in least busy area
139 set key top right
140
141 # Title and axis labels
142 set title "CPU and Memory Usage During Dual Attack" # change title
      accordingly
```

```
143 set xlabel "Time in Seconds"
144 set ylabel "Memory%"
145 set y2label "CPU%"
146
147
148 plot "status.dat" using 1:2 axes x1y1 title 'Memory' with linespoints
        pointsize 0.5 pointtype 3,"" u 1:3 axes x1y2 title 'CPU' w
        linespoints ps 0.5 pointtype 7
149
150 EOFMarker
151
152 }
153
154 function main() {
155   mem_cpu
156   gnu_plot
157 }
158 main
```

Appendix F

Snort Eth0 configurations

```
#copy original snort.conf file
Sudo cp /etc/snort/snort.conf snort_eth0.conf
sudo nano /etc/snort/snort_eth0.conf
#configurations start from here
ipvar HOME_NET 192.168.10.0/24
ipvar EXTRNAL_NET !$HOME_NET
var RULE_PATH /etc/snort/rules
var SO_RULE_PATH /etc/snort/so_rules
var PREPROC_RULE_PATH /etc/snort/preproc_rules
var WHITE_LIST_PATH /etc/snort/rules
var BLACK_LIST_PATH /etc/snort/rules
config pcre_match_limit: 3500
config pcre_match_limit_recursion: 1500
config detection: search-method ac-bnfa-nq search-optimize max-pattern-
    len 20
preprocessor perfmonitor:
   time 300 events atexitonly flow max console pktcnt 10000
output log_tcpdump: /var/log/snort/eth0/snort.log
include $RULE_PATH/eth0.rules
```

Appendix G

Snort Eth0 Sample Rules

```
# DoS on port 80 rule
alert tcp $EXTERNAL_NET any -> $HOME_NET 80 (msg:"TCP SYN flood attack
    detected on Port 80"; flow: stateless; flags:S,12; threshold: type
    threshold, track by_dst, count 1000, seconds 5; classtype: attempted
    -recon; sid:10002; rev:1;)
# Rule for detecting BlackNurse attack
alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"TDC-SOC-Possible
    BlackNurse attack from external source"; itype:3; icode:3;
    detection_filter:track by_dst,count 250, seconds 1; priority:3; sid
    :88000012; rev:1;)

# Telnet/Botnet traffic detection rule
alert tcp $EXTERNAL_NET any -> $HOME_NET [23,2323,3323,4323] (msg:"ET
    TELNET busybox MIRAI hackers - Possible Brute Force Attack"; flow:
    to_server,established; content:"MIRAI"; flowbits:isset,ET.telnet.
    busybox; threshold: type limit, count 1, track by_src, seconds 30;
    reference:url,lists.emergingthreats.net/pipermail/emerging-sigs
    /2016-August/027524.html; classtype:attempted-admin; sid:2023019;
    rev:2; metadata:attack_target Server, deployment Datacenter,
    signature_severity Major, created_at 2016_08_08, performance_impact
    Low, updated_at 2016_09_26;)
# Port scan rule i.e. XMAS Tree scan
alert tcp any any -> $HOME_NET any (msg:"Nmap XMAS Tree Scan"; flags:FPU;
     sid:10000006; rev:1; )
# SSH dictionary attacks
alert tcp $EXTERNAL_NET any -> $HOME_NET 22 (msg:"INDICATOR-SCAN SSH
    brute force login attempt"; flow:to_server,established; content:"SSH
    -"; depth:4; detection_filter:track by_src, threshold: type
    threshold, count 10, seconds 5;metadata:service ssh; classtype:misc-
    activity; sid:19559; rev:5;)
```