2019

A System to detect suspicious activities in network traffic

Roselyne M. Gesare
Faculty of Information Technology (FIT)
Strathmore University

A System To Detect Suspicious Activities In Network Traffic

Magangi Roselyne Gesare

Submitted in total fulfilment of the requirements for the Degree of
Master of Science in Information Technology at Strathmore University

Faculty of Information Technology
Strathmore University
Nairobi, Kenya

June, 2019

Declaration

I declare that this work has not been previously submitted and approved for the award of a degree by this or any other University. To the best of my knowledge and belief, the proposal contains no material previously published or written by another person except where due reference is made in the proposal itself.

Magangi Roselyne Gesare

Signature:……

Date: ……

Approval

This thesis of Magangi Roselyne Gesare was reviewed and approved by the following:

Dr. Vincent Omwenga,
Senior Lecturer, Faculty of Information Technology,
Strathmore University

Dr. Joseph Orero,
Dean, Faculty of Information Technology,
Strathmore University

Prof. Ruth Kiraka,
Dean, School of Graduate Studies,
Strathmore University

Abstract

Modern enterprise networks have become targets of attacks from Internet malware including worms, self-propagating bots, spamming bots, client-side infects (drive-by downloads) and phishing attacks. The results of a cyber-attack which include loss of company information, theft of money, costs of repairing the affected systems and perhaps damage to the reputation of the organization, can be devastating. However, with the right tools, security can dissect suspicious traffic to detect these attacks.

When a company institutes a good method of network security surveillance, security analysts could be alerted within minutes of problems occurring in good time. It is with this aim that this study sought to research and develop a simple and robust system that could be used to detect suspicious activities in network traffic. Specifically, the study sought to; Discuss and analyze suspicious activities in network traffic and devices; analyze the existing techniques used to detect suspicious activities in network traffic; develop a system for detecting suspicious activities in a network traffic; and validate the proposed system. The study adopted an experimental design.

The experiment was conducted on an Ubuntu machine running 16.04 LTS where Snort was installed alongside PulledPork, Barnyard2 and BASE to act as the Web GUI. ICMP large packets were sent to the network for detection and the system was able to detect, analyze and report them on the BASE GUI.

The target population for this study was network traffic. The researcher generated the network traffic through sending data packets across the networks. The network traffic was analysed by using the network security tools analysed by the researcher and chosen based on their availability and compatibility with one another to come with the desired setup.

This research was not aimed at reinventing the wheel but offering major improvement through precise feedback on what network administrators across different organizations could identify as suspicious activities in their networks.

Keywords: **Snort; Basic Analysis and Security Engine(BASE), PulledPork; Barnyard2; Internet Control Message Protocol (ICMP).**

Table of Contents

List of figures

List of Tables

Abbreviations and acronyms

ADS	–	Anomaly Detection Systems
APT	–	Advanced Persistent Threats
AV	–	Antivirus
DB	-	Database
DoS	–	Denial-of-Service
HIDS	-	Host intrusion detection systems
ICMP	–	Internet Control Message Protocol
IoT	–	Internet of Things
IDS	–	Intrusion Detection Systems
IPS	–	Intrusion Prevention System
KE-CIRT-CC	–	Kenya National Computer Incident Response Team Coordination Centre
MIB	–	Management Information Base
NIDS	–	Network Intrusion Detection Systems
PC	–	Personal Computer
SNMP	–	Simple Network Management Protocol

Definition of terms

Suspicious activity – Activity that is out of the ordinary (Silowash *et al.*, 2012). Njemanze and Kothari (2008) pose that all inbound and outbound network activity should be inspected and identified as suspicious when necessary when network or system activity indicates that someone is attempting to break in.

Network traffic – refers to the amount of data moving across a network at a given point of time (Network dictionary, 2007).

Acknowledgement

Primarily I would like to thank God for giving me the strength, knowledge and sustaining me until the end. The completion of this project would not have been possible without the guidance and consultation from my supervisor Dr. Vincent Omwenga**.** In addition, I would like to appreciate the encouragement, support and constructive criticism and corrections from my classmates especially Humphrey. This Thesis is from sheer hard work and determination. I am very grateful to them for their tolerance and patience throughout the progression of the Thesis.

Dedication

I dedicate and owe considerable thanks to my Husband Lewis Osero for encouraging and supporting me throughout my studies.

Chapter 1

Introduction

1.1 Background to the Study

Due to the persistent and universal nature of the Internet's traffic sent to unused addresses, modern enterprise networks have become continuous targets of attacks from a large number of attacks from Internet malware including worms, self-propagating bots, spamming bots, client-side infects (drive-by downloads) and phishing attacks (Pang *et al.*, 2005). According to Zhu, Yegneswaran and Chen (2009), estimates on the number of malware instances released vary vastly (between tens of thousands to more than hundreds of thousands per month) depending on census methodologies. However, there is consensus that malware is becoming increasingly widespread, sophisticated, and a formidable threat not just to the network communications but also as a purveyor of data and identity theft (Chen, Dimitriou & Zhou, 2009).

The widespread growth of networks has created remarkable possibilities and a big challenge for people and organizations. The uncontrolled growth of networks has also led to an increase number of intruders on the internet making everyone to be at risk of attacks. Therefore, a strong mechanism is a necessity to protect computer systems and organizations from intrusion. Safwan (2016)

Spiegel, McCorkendale and Sobel (2007) noted that a network worm automatically spreads from computer to computer by exploiting software vulnerability that allows an arbitrary program to be executed without proper authorization. For instance, Acohido and Swartz (2004) reported that the Sasser worm located a Personal Computer (PC) running a vulnerable operating system and successfully compromised the machine in less than four minutes from when the machine was connected to the internet. The Slammer worm, which broke into the majority of vulnerable hosts on the internet in less than ten minutes, congested many networks and left at least 75,000 hosts infected (Moore & Shannon, 2002).

According to Mitrokotsa and Douligeris (2007), once a host has been compromised, it can be used for such heinous activities such as launching Denial-of-Service (DoS) attacks, relaying spam emails, and initiating the propagation of new worms. An example is the Bagle worm that downloads a program that turns an infected machine into a zombie that an attacker can control

remotely (Czosseck & Geers, 2009). As a result, a large number of exploitable machines are readily available to an attacker, thus facilitating distributed and large-scale automated attacks. Furthermore, because attackers tend to exploit a vulnerability soon after it becomes known, it is extremely difficult for system administrators to patch every vulnerability on their machines before a new malicious code is released. The speed and prevalence of automated attacks render ineffective any legacy defences that rely on the manual inspection of each case. It is necessary to deploy an automated defence system that continuously monitors network traffic, determines whether the traffic from a particular source reveals a certain malicious activity, and then triggers an alarm when it finds such traffic (Costa, 2007; Jung, 2006). Furthermore, it is more important if the defence system makes it easier to analyse these attacks and provide a good summary for security and network professionals to detect situations out of the normal operation.

1.1.1 Global Perspective on suspicious activities

The global perspective of suspicious activity is not very different from the Kenyan perspective discussed below. A report presented before the U.S. House of Representatives, Committee on Financial Services, Subcommittee on Oversight and Investigations in June 2015 further explains the extent to which cyber-attacks around the world is taking shape. The report approximated that an attack happens every 34 seconds every day. These are mainly targeted at banks and financial institutions. The interesting bit is that the rate of an attack every 34 seconds is not an attack that banks are aware of. They are somewhat new attacks or attempts to breach the system (J. Cilluffo, Madon & Bejtlich, 2015).

As J. Cilluffo, Madon and Bejtlich (2015) explain, attacks no longer target the big businesses that can be termed as wall street listed businesses but small and medium sized businesses. The report supports that a need for mitigation and detection is important. As much as companies are trying to create a good defence system, the advancement of technology and the ubiquitous nature that it comes with pushes innovation higher and at the same time, widens the spectrum and complexity of the nature of attacks around the world.

The global perspective attacks include a number of actors who act on different intentions, motives and capabilities. Beyond this, terrorists pose a threat because they usually have the right intentions to launch attacks but lag behind in terms of capabilities. Criminal organizations on the

other hand prove to be capable but their motivation and intent is different from what drives terrorists.

1.1.2 Kenya Perspective on suspicious activities

Cherono (2017) reported that a wave of unprecedented cyber-attacks had swept across the globe, with over 350 companies and hundreds of thousands of computers in 152 countries affected. The attack by a computer worm or ransomware called WannaCry (Wanna Decryptor) targets the Microsoft Windows operating system, encrypts files and demands that the user pay ransom before being allowed to continue using the computer.

In Kenya, computer forensics and data recovery company East Africa Data Handlers said it had received 14 cases of servers that had been affected by the ransomware. Among these clients, two were multinationals, who had the entire 15-year data manipulated and lost. The existence of the malware in the country was confirmed by the country's cyber security response agency, the National Kenya Computer Incident Response Team Coordination Centre (KE-CIRT-CC) managing director, Mr. George Njoroge of the East Africa Data Handlers. He warned that many companies, in the country were at risk of attacks by the ransomware because many companies and individuals do not upgrade their security infrastructure, mostly because of the current economic challenges. He added that the best solution was to keep pace with the dynamic changes in technology (Cherono, 2017).

According to Serianu (2016), there was a growing dependence on third parties by organizations in Kenya which had resulted in introduction of new attack vectors. Organizations were handing over a lot of the typical controls that you would expect to see internally to these third parties. Kenyan organizations were not adequately performing risk assessment on their service providers before or during their engagements. As a result, many breaches that occurred in the recent past involved a third party in one way or another. Further, the report pointed out that the Internet of Things (IoT) or Internet-connected devices were growing at an exponential rate and so were related threats. IoT had been adopted into various sectors of the economy including agriculture, healthcare, energy and transportation. Due to their insecure implementation and configuration, these Internet-connected embedded devices, including CCTVs and nanny cams, Smart TVs, DVRs, Smart routers and printers, were routinely being hacked and used as weapons in cyber-attacks.

1.2 Statement of the Problem

Most organizations use computers connected on a network to streamline their business process (Fuchsberger, 2005). While they connect to the internet, their Information Technology systems are always under the risk of attacks from intruders. Some of these attacks are sophisticated and hidden which result to loss of company data, theft of money, costs of repairing the affected systems, fines and regulatory sanctions and perhaps worst of all, reputation of the organization.

Network security surveillance and host monitoring is one of the best ways to detect modern threats attacking a networked environment. With the right tools, security can dissect suspicious traffic and analyse suspicious activities to detect these attacks. When a company institutes a good method of networked environment security surveillance, security analysts can be alerted within minutes of problems occurring in good time. However, Kohn, Eloff and Eloff (2013) note that, while it is standard practice to examine logs to discover or conduct forensic analysis of suspicious activity in a network, such investigation remains a largely manual process and often relies on signatures of known threats. A major challenge is that security products often come from a patchwork of vendors and are inconsistently installed and administered. They produce log data with widely differing formats, and logs that are often incomplete, mutually contradictory, very large in volume (e.g., security-relevant logs in a large enterprise grow by terabytes per day), and filled with non-specific or invalid event records. Besides the logs from numerous packets capturing software being complicated, viewing them is a problem. It requires one to be keen and read a lot of unnecessary information while at it.

Many techniques have been adopted yet most organizations' servers, websites and personal accounts are still being attacked. Intrusion prevention systems tend to attempt to stop an intrusion attempt but at the same time, it is not able to see the bad traffic that lies within a packet that is assumed to be good. Signature-based Intrusion Detection Systems (IDS) can easily detect known attacks but are unable to detect new attacks for which there is no pattern available. Anomaly Detection Systems (ADS) detect unknown attacks by comparing new behaviour against a trustworthy activity. This approach enables the detection of previously unknown attacks but the risk of legitimate activity being classified as malicious is increased. This indicates a gap on the most effective technique to be integrated to effectively counter these challenges. In

order to fill this gap, this study proposed the development of an efficient, customizable intrusion detection system that is a mixture of signature based, behaviour based and intrusion prevention system to have high accuracy detection rate and reduction of false alarm rate.

1.3 Objectives of the Study

1.3.1 General Objective

The purpose of this study was to develop a system that can be used to detect suspicious activities in the network traffic.

1.3.2 Specific Objectives

i. To discuss suspicious activities that can be found in network traffic.
ii. To review the existing techniques used to detect suspicious activities in network traffic.
iii. To develop a system for detecting suspicious activities in a network traffic.
iv. To test the system for the detection of suspicious network activities.

1.4 Research questions

i. What suspicious activities can be found in network traffic?
ii. What are the existing techniques used to detect suspicious activities in network traffic?
iii. How can a system for detecting suspicious activities in network traffic be developed?
iv. How can the system for detecting suspicious network activities be tested?

1.5 Justification for the Study

Every organization needs to protect its assets. The assets mean the information that is stored in the computer networks, which is as crucial and valuable as the tangible assets of the company. (Courtney, 2014). This hence calls for every person and organization on the network to monitor their system for unauthorized access and any other attacks or suspicious activities. Some of these attacks are sophisticated and hidden which may result to loss of company data, theft of money and perhaps worst of all, reputation of the organization.

It is therefore important to deploy an automated defense system that continuously monitors network traffic, determines whether the traffic from a particular source reveals a certain malicious activity, and then triggers an alarm when it finds such traffic (Costa, 2007; Jung,

2006). Besides that, the system should present an easy to use and analyze interface that presents the network activities to network administrators and security analysts. The network administrator is then able to get the data secured by taking appropriate action against the attacks. Interfacing between the logs and a good UI system has been a problem especially when customizing IDS.

Tremendous growth and usage of the internet al.so raises concerns about how to protect and communicate the digital information in a safer manner. This prompts for a review of the systems and techniques used.

1.6 Scope of the study

The study provided sufficient insight on detection of suspicious activities on network traffic based on intrusion detection: signature based and anomaly detection and intrusion prevention systems. The work done by Wu et al. (2008) explains that signature-based intrusion systems analyze information it gathers and compares it to a large database of attack signatures. The system looks for a specific attack that has already been documented. Snort is one of the main softwares used as a packet sniffer and a logger. Integrating snort with BASE provided a web front-end query and analyzed the alerts coming from a snort system. Barnyard2 was one of the additional components, which processed the incoming packets and save them in MySQL database. Pulled pork downloaded automatically the latest snort rules.

This research focused on looking to sniff out ICMP traffic from the network, this will be done by SNORT. The module that will implement this functionality was written by the researcher using custom rules. The researcher opted to only alert whenever and ICMP packet is sent on the network and on this, a web UI was used to show the alerts.

Chapter 2

Literature Review

2.1 Introduction

This chapter exhaustively looks at the previous work in the field of Network Security. The chapter starts by scrutinizing suspicious activities in network traffic. Where the chapter categorizes them into suspicious activities within the LAN and WAN then followed by a thorough discussion on DDoS. This is followed by an analysis on the existing techniques used to detect suspicious activities in network traffic. The chapter looks at the existing implementation of systems for detecting suspicious activities in network traffic that has been developed. Finally, the concept diagram of this research is presented.

2.2 A discussion of suspicious activities in network traffic

Analyzing network traffic as suggested by (NEEDHAM & Lampson, 2008) show some clear resemblance between the attacks that malicious users propagate now and the attacks that were being propagated in the early 2000s. NEEDHAM and Lampson (2014) go ahead to discuss that as much as these attacks leverage on the vulnerabilities of network protocols and software, patches and updates are being released day in day out. The same way, attackers are still finding similarities in these bugs that make them just improve on their attack procedures and are able to infiltrate networks and systems.

This paper acknowledges that IP is the most popular networking protocol that is used by most devices in a networked environment. Owing to the nature of operation of the IP protocol where it does not provide authentication services or confidentiality, most attacks can emanate from this vulnerability and compromise the network or the systems in the network. These attacks can be launched from within the LAN or be within the WAN.

2.2.1 Attacks on Local Area Networks

Individuals within an organization can propagate such attacks or suspicious activities by taking over an account using someone else's credentials and committing fraud. How such activities are conducted involves the deployment of a packet sniffer that harvests the passwords of the root

accounts and use them to create other fraudulent accounts to perpetrate fraud on the information systems. Such activities are quite hard to detect and require some form of thorough network traffic and host activities analysis to notice the suspicious activities that occur in the network.

Another sophisticated attack or suspicious activity in a LAN is hijacking ARP messages and giving out wrong responses to client computers hence an attacker pauses as the victim. This kind of attack requires the victim machine to be powered off so that they cannot be alerted when this theft of identity happens (Vidya et al., 2011). However, attackers are patient enough to execute this. In other instances, the attacker can decide to launch another attack that forcefully brings down the victim's machine. This can be done by throwing the machine off the network by assigning subnet masks that are out of scope whenever they are joining the network after a cold boot or a soft boot. As for the victim's machine, their layer 3 address makes them vanish in their subnet thus become invisible. This makes it possible for the attacker to pose as the victim.

Such sophisticated attacks on identity theft cannot be easily detected by running packet sniffer software or log analysis from a naïve approach. It is incumbent on security analysts to customize the network security tools to notice these subtle changes in their network activity and possibly notify them early enough as (Vidya et al., 2011) explains.

2.2.2 Attacks that take advantage of WAN connections

With IPv4 lacking the authentication and confidentiality capabilities other supporting technologies that make communication possible like TCP suffer from compromise. The SYN flooding attack that study the three-way handshake procedure of the TCP and then floods an unsuspecting user with multiple SYN messages. Probably more than they can handle that hence leaves them with half-open connections and eventually make them unable to run any other service (Stallings et al., 2012).

Smurf attacks are among the unusual network activities that happen in the background and end up utilizing a network bandwidth without the administrator noticing. Propagation of this attack happens when the attacker sends out numerous ICMP requests and forges the source address to be a victim's computer in their target network. These ICMP echo requests can hit other Smurf amplifies as identified by NEEDHAM and Lampson (2008) which quotes the website www.netscan.org to be known to propagate Smurf amplifiers. The machines that receive this

ICMP echo request will respond, if they are alive to victim's computer and swamp the target with more ICMP packets that it cannot cope with and eventually bring them down. Servers are the target for these kinds of attacks.

Source level routing, a concept that was introduced in TCP to get around bad routers had its own vulnerabilities that attackers would leverage on to reroute their target computers to send packets to the wrong intermediary devices. Hence, they end up being manipulated. Message redirecting in networks is also an activity that goes on silently without the administrator noticing there is an attacker, effectively saying, as NEEDHAM and Lampson (2008) states, "You should have sent this message to the other gateway instead". This redirection goes on in a network without checking and most ISPs and other supporting services like mail routing become less competent to not allow source routing and redirection of packets from host machines. This poses the danger to intercept packets and modify them or make them disappear.

2.2.3 The broad umbrella of DDoS attacks

DDoS attacks are an extrapolation of Denial of Service attacks (Visbal, 2015). As much as DoS prevents legitimate users from using a system, DDoS will do the same thing but the method of deployment is what sets these attacks apart. A Distributed Denial of Service (DDoS) attack is a coordinated attack on the availability of services of a given target system or network that is launched indirectly through many compromised computing systems (Specht & Lee, 2004). The services under attack is usually called the "primary victim", and the compromised systems that attackers use to launch the attacks is called the "secondary victims." The use of secondary victims in a DDoS attack places the attacker at a vantage position because the effects of their disruption of network operations can scale far and wide but they will always remain anonymous. This makes it more difficult for network forensics to track down the real attacker.

The primary and secondary victims in a DDoS attack represent the most common model of launching an attack(Bhuyan et al., 2014). The agent handler model comprises of an attacker, the handler, the agents and the victims. The attacker communicates with the handlers who are widespread within the internet. The handler then identifies the agents that they can compromise or have been compromised already and use them to launch an attack on the victim.

The taxonomy of DDoS attacks is what the broad area of security dissects and coin terms to describe specific attacks. Some of these attacks were discovered in the early years and have since been improving while others are recent developments that respond to the growing transformation of the internet and protocols that run it.

However, DDoS attacks can be broadly categorized into these two; bandwidth depletion attacks and resource depletion. From these, the sub categories of the attacks can emanate. Bandwidth depletion attacks can be either amplification attacks that involves zombie computers sending messages to a broadcast IP address causing all the devices in the subnet reached by the broadcast address to send a reply to the victim system (Visbal, 2015).

The other bandwidth depletion attack method can be flood attacks. Ahn and Jang (2012) supported by Specht & Lee (2004) explain and describe flood attacks as zombies sending large volumes of traffic to a victim's system causing the bandwidth to be congested and making the system being inaccessible by legitimate uses. The flood attacks can be UDP flood attacks which send numerous UDP requests to a victim's computer. The UDP requests can be sent to a random or specified port. The victim will try to process these requests and render it to the correct application. If it does not have an application that supports the request, the victim will automatically send out an ICMP packet indicating that the destination port is unreachable.

Resource depletion attacks on the other hand involve an attacker sending packets that misuse network protocol communications or are malformed (Specht & Lee, 2004). Network resources are tied up so that none are left for legitimate users. The most common resource depletion attack is the TCP SYN attack that this paper has described earlier. The other resource depletion DDoS attack is the misuse of the PUSH+ACK protocol. In a PUSH + ACK attack, the attacking agents send TCP packets with the PUSH and ACK bits set to one in the packet header information. These will trigger the victim's system to unload all data in the TCP buffer (regardless of whether or not the buffer is full) and send an acknowledgement when complete (Specht & Lee, 2004). If this process is repeated with multiple agents, the receiving system cannot process the large volume of incoming packets and the victim's system will crash.

2.3 Existing techniques used to detect suspicious activities in network traffic

An anomaly detection technique in networks refers to the problem of finding exceptional patterns in network traffic that do not conform to the expected normal behaviour. These nonconforming patterns are often referred to as anomalies, outliers, exceptions, aberrations, surprises (Bhuyan et al., 2014). This anomaly detection in a network environment is a wide growing field that has gained momentum of late with the concomitant rise in the adoption of machine learning and statistics when developing tools to detect anomalies in the network (Bhuyan et al., 2014). Detection of anomalies is moving from the manual process where the security analysts are required to continuously audit network activities and make sound judgments when detecting network outliers.

Network anomaly detection techniques that Bhuyan et al. (2014) broadly discusses in his work can be categorized into four classes. They are statistical, classification-based, clustering and outlier-based. A statistical method usually bases its aberration detection procedure by looking at activities from a given data set that have a low probability of being generated. Applying statistical inferences, the technique can then decide if the activity belongs to the statistics model or not. If it does not, it is automatically flagged off as a network anomaly.

Classification on the other hand looks at categorizing network activities into classes and then matching each of the incoming and outgoing network activities in the existing classes. The classification of the network activity is based on the availability of a training dataset that categorizes the membership of activities in network traffic. Clustering is assigning a set of objects into groups called clusters. Objects in the same cluster have more similarities in some sense compared to objects that are in different clusters. Clustering is used in explorative data mining techniques. Outliers are those points in a dataset that are highly unlikely to occur given a model of the data. Clustering and outlier finding are examples of unsupervised machine learning as Bhuyan et al. (2014) states. Bhuyan et al. (2014) go further and confirms that clustering can be performed in network anomaly detection in an offline environment.

Another method that have been used to detect suspicious activities is signature-based filtering of traffic in Intrusion Prevention Systems. Signature based filtering mechanism in network IPS and IDS borrows the concept from the operation of legacy and current antivirus. A signature-based detection mechanism can be termed to be a reactive nature of detecting suspicious activities. The

nature of operation of a signature-based system is based on looking at the pre-recorded patterns and flagging out patterns that match to its records. This method is very sufficient in detecting known attacks. However, it becomes challenging when we are faced with new attacks that keep changing form and pattern in networked environment.

2.4 System developed for detecting suspicious activities in network traffic

Techniques that have been discussed above have been deployed in most IDS to try and combat suspicious activities which can either be anomalies in a network or not. They have provided network administrators and security analysts with an easy task of securing their networks and having solid explanations to the activities that go on in their networks. Most of these IDS borrow from the techniques discussed in the section above to create a system that can detect network attacks (suspicious activities).

Minnesota Intrusion Detection System (MINDS) employs the use of data mining to detect network intrusion (Ertoz et al., 2005). It collects data through flow tools that act as its input, the system then looks at the packet header information and build a one-way session to the flows. MINDS work hand in hand with NetFlow which is the data collection tool. MINDS use the data in batch mode since it is presented with a lot of data to analyze and make decisions. However, not all data will be sent to MINDS for analysis. Based on the use specifications, the data of interest will be sent to MINDS engine for analysis (Bhuyan et al., 2014).

The first step that MINDS takes, following the discussion Bhuyan et al. (2014) have is to extract the important features that it wants to use based on a summary called time windows. The known attack detection module is the next feature to be fired up. It is used to detect network connections that correspond to attacks for which signatures are available, and to remove them from further analysis.

Next, an outlier technique is activated to assign an anomaly score to each network connection. A human analyst will then look at only the most out of normal connections to determine if they are smoke screens or actual abnormal behaviour in the network.

Integrating Elastic Search and Kibana (ELK stack) and packet capturing application like Wireshark in doing real time network analysis as described by the project by Elastic Blog (2018) is also a good way to detect suspicious network activities using an integrated system. Elastic

search from Kuc et al. (2015) and Owuor (2016) is an open source search engine library built on Apache Lucene. It is written in Java and uses Lucene internally for all of its indexing and searching. On the other hand, it is a log shipper and log manager. It is a tool for managing events and logs. Owuor (2016) explains that it is written in JRuby and is easy to deploy since a single JAR file can be started directly from the CMD prompt or the terminal. An agent can act as a shipper which sends and collects events to another instance, an indexer which receives and indexes the events, a searcher storage, which searches, and stores events or a web interface depending on the configuration file.

Kibana is a web interface for Elasticsearch. It is used to give the robust GUI that network analysts and security engineers need to visualize the data stored in the Elasticsearch database (Gupta, 2015). It displays data in form of area charts, data tables, line charts, markdown widgets, metrics, pie charts or tile maps. The tool is used to provide the analysis options of the centralized logs. Gupta (2015) further explains that Kibana requires a webserver and it is written in HTML and JavaScript language.

Integrating the ELK stack means that network security experts will be able to write customs rules on before it ships the logs or the packets it has captured from Wireshark to Elastic search. Looking at the nature of the attacks from DDoS, resource depletion attacks or bandwidth depletion attacks, it can be easy from customizing to identify the handlers in such a system and shut them down.

2.5 Conceptual Framework

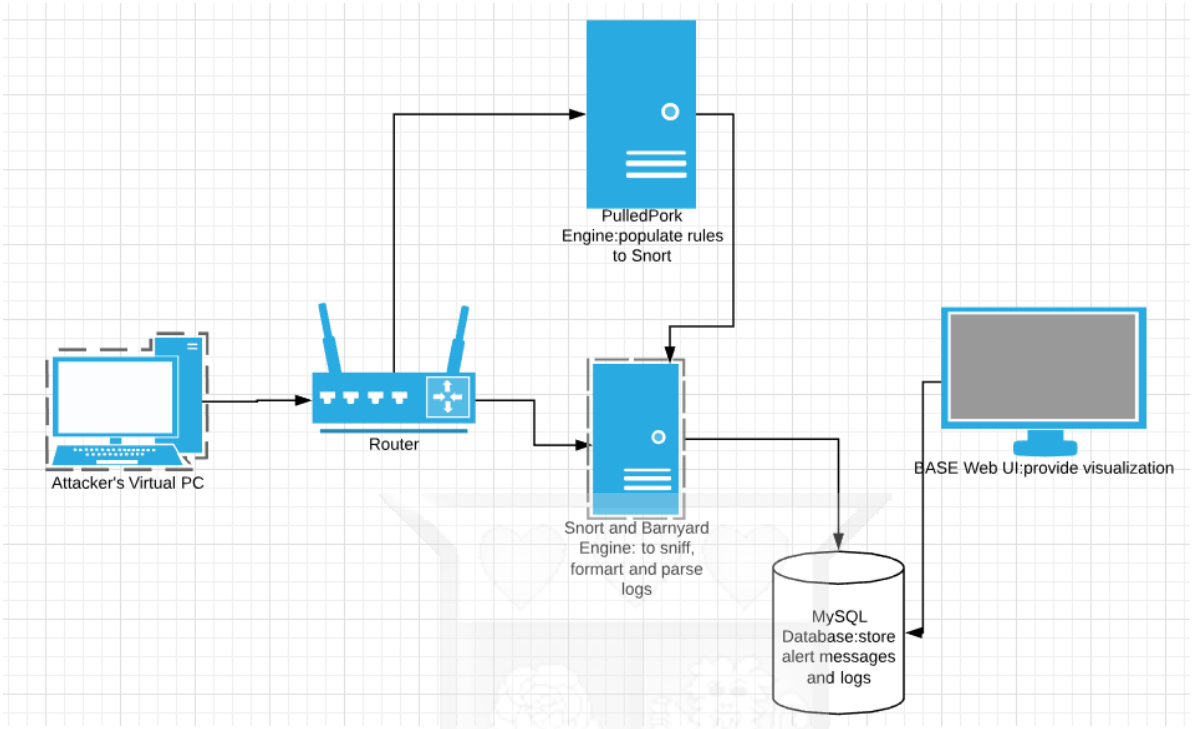


Figure 2-1: System concept diagram

The concept diagram captured in figure 2-1 above represents the solution that this research sought out to implement. From the discussion in section 2.4 above on the existing systems that have been developed to detect suspicious activities in network traffic, one thing stands out. For better detection, three system components need to merged. Rules need to be written and saved somewhere, hence later applied to check against input to the system which will be network traffic. This rule and input matching procedure will produce some logs. These logs, in most systems like Gupta's (2015) and Owuor's (2016) utilized the ELK stack to ship the logs. The logs were shipped and stored using Logstash and elastic search respectively. Eventually, the last module was a visualization tool which was Kibana as Gupta's (2015) and Owuor's (2016) explain for the ELK instance.

Using the same approach, this research, through the deployment of Snort and Barnyard2 engine as shown the diagram 2-1 above aimed at preforming rule matching against traffic that was coming from the attacker's network through the router. Rules were provided the Snort engine using PulledPork. Once logs have been generated using the rule matching procedure, they were

parsed and sorted in the MySQL databased using Barnyard2. As in the case for ELK, using Kibana as the visualization tool and Elastic search as the database, this research used BASE as the visualization tool to fetch data from the logs stored in the MySQL databased and display them on the Web GUI housed by BASE.

Chapter 3

Methodology

3.1 Introduction

The chapter looks at the approach that the research took to ensure the work proposed produced a reliable system that can detect suspicious activities in a networked environment. System Development Methodology kicked off the scope of this chapter followed by the Research Design.

Data collection methodology that was used in the research and the sample space is discussed in this chapter. The chapter then proceeds to look at how data was collected and analyzed to suit the needs and answer the questions presented by this study. Finally, the chapter justifies the quality of the research and looks at the ethical considerations made when conducting the research.

3.2 System Development Methodology

This research was based on test-driven deployment. According to Astels (2003), this is a commonly-used development methodology by which (failing) tests are initially created, and only then is the actual software code created, which aims to pass the newly-generated tests. The researcher chose this approach owing to the nature of the system to be developed. Since the research was dealing with different components and integrating them, numerous tests were conducted whenever one component was built and configured. Numerous follow up test were conducted again after the integration phases was complete. This was to make sure that the initial tests being done maintained the status quo and allowed the researcher to move on to the next step.

The researcher developed and wrote a test for Snort tried and tested Snort in the LAN. On numerous occasions, the researcher had to edit configuration files for Snort and check if the changes had been affected. Afterwards, Snort as a module was kept aside and other supporting components like Barnyard2 and PulledPork had to be developed using a test-driven approach as highlighted earlier. At the integration phase, modification to the configuration files were made to make sure Barnyard2 and PulledPork assisted Snort to sniff, parse and check if the rules fetched by PulledPork were conformed to by the traffic in the network. At each stage, a test was conducted, if the expected result was not achieved, the researcher had to debug the configuration

file or check the module logs to come up with a solution. The same approach was taken when transporting the logs to the MySQL database for storage, tests were conducted to make sure MySQL DB was correctly set up and was running on the Linux testbed. In addition, an iterative process of making sure Snort, PulledPork and Barnyard2 worked together to generate the data needed to be pushed to the DB. This integration with the database failed a couple of times. The system and error logs generated from this enabled the researcher to successfully debug the system and finally deploy. The last phase of integrating the web GUI was dependent on other modules. For BASE to run, the researcher had to install it correctly and make sure it is compatible with the LAMP stack that already existed. This called for a downgrade of PHP and an installation of Perl throughout the test and deployment stages to make sure BASE was up and running. For final testing, the researcher had to bring in together Snort, Barnyard2 and PulledPork and make sure they communicated to BASE.

However, as Beck (2003) notes, this methodology allowed the system developer to always come back and make changes whenever they experience a failed test in future to enhance the completion of a working project. The steps that are involved in the test-driven methodology are writing a test, confirming the test fails, writing system to pass test, confirming the test passes, refactor and repeating all the steps in the existing components. As described above, the researcher found this approach to be effective. The figure 3-2 illustrates the methodology used in detail.

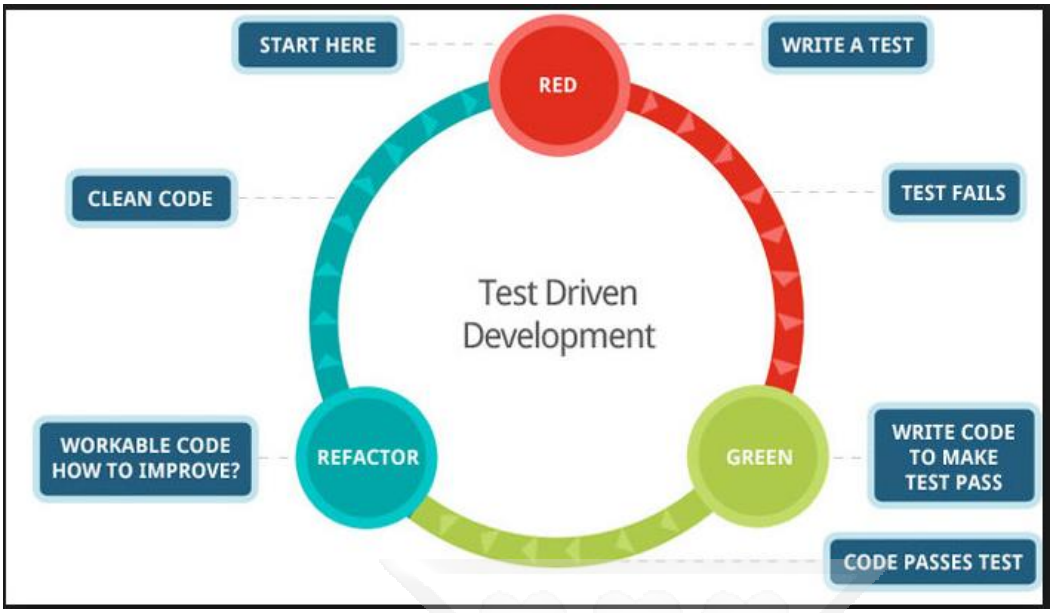


Figure 3 -2: Test driven methodology diagram (adopted from Anon, 2019)

3.3 Research Design

For this research to succeed, both qualitative and quantitative methods were used. A review of multiple literature was done to find out the major suspicious activities that exists in network traffic. This formed part of the qualitative approach that this research took to achieve its objectives.

However, for completion and materialization of the research, a scientific approach was adopted. This backed up the quantitative approach because primary data was collected to aid in the analysis of the deployed system and description of the obtained results from the experiment conducted. Beyond the deployment and collection of primary data, the research also compared the findings it gathered to the existing field of knowledge on network attacks and deployment of NIDS.

This thesis documentation leveraged on experiments carried out from time to time to ensure that what is classified as a suspicious activity was flagged out. In this case, ICMP echoes and replies were used for demonstration. This expounds on the scientific approach the research took. This called for multiple, repetitive experiments to be conducted. In the same vein, instances of test-driven deployment were used to develop the system. This is owed to the fact that systems parameters kept changing. New information also came up during the system development phase

like downloading and updating rules from Barnyard2 and PulledPork that led to numerous adjustments being made here and there to ensure the system met the objective of the research.

3.4 Target population and sampling frame

The target population for this study was network traffic. The researcher generated the network traffic through sending data packets across the networks. The network traffic was analysed by using the network security tools, BASE, PulledPork, Snort and Banyard2. These tools were chosen based on their availability and compatibility with one another to come with the desired setup. Therefore, the sampling technique that the researcher settled for was convenient sampling based on the nature of the type of attack launched and tools available.

3.5 Data collection

Primary data was collected to allow the researcher document a report on how well the system was able to meet the objectives that were set out in the first chapter. Data from the BASE web user interface indicating that ICMP traffic was being flagged out by the system formed the primary data that this research collected. This was collected from the experiment setup that the researcher proposed and actualized in the next chapter.

This data was collected mainly through observation of the results that were displayed by the Web GUI. This primary data collected, for it to make sense to the research and prove its a liability as the expected outcome, the researcher compared it to the existing knowledge in the field of information and network security to see if the system was able to enhance knowledge in the area and conformed to it.

3.6 Data analysis and presentation

Data collection methods influenced the analysis that the researcher engaged in while deploying and testing the system. Primary data was collected on the different terminal screens in an iterative fashion whenever the researcher was working on the different components of the integrated security system for reporting malicious activities. Other data was collected from the web browser where BASE was running to collect data and present. Hence, analysis and judgment were based on this primary data coupled with secondary data concepts. This research relied highly on descriptive analysis to demonstrate how it met the objectives during the system development and testing phase. Images and accompanying explanations and descriptions were

used to analyze the data collected by the researcher. Analysis from the observation was used to describe the images.

3.7 Research Quality

The research sought to ensure detection on suspicious activities in a networked environment borrows from the best practices used by previous works that have gone into the deployment of SNORT and BASE.

This research was not aimed at reinventing the wheel but at offering major improvement through precise feedback on what network administrators across different organization can identify as suspicious activities in their networks. Besides that, the research wanted to make this process seamless by interfacing an easy to use web GUI to aide in this process. This meant that harmonization of old and new knowledge was key. Integration of this, with new ideas was seen to be ideal.

3.8 Ethical considerations

The research was deployed in multiple network environments where sample attacks were being launched to test the functionality of the system. This research did not target the attacks to any other machine other than the ones documented by the researcher that was used for testing purposes. Besides that, the researcher did not plug in the integrated security system to scan any network subnet that its Snort engine was not operating on. In any case the researcher was to note any suspicious activity against their own subnet to a machine that is not among the test machines. The researcher was determined to advice and notify the affected party.

Chapter 4

System Analysis and Design

4.1 Introduction

This chapter elaborates on the steps taken by the researcher during the design phase of the system. The chapter starts by looking at the requirement analysis, which encompasses a discussion of the functional and non-functional requirements the system set out to meet. The chapter analyses the optimal environment for setting up the system for detecting suspicious activities in network traffic. This discussion is backed up by the analysis of the security engines the researcher used, Snort and Base. Finally, the research outlines the systems operations in a series of use case and sequence diagrams.

4.2 Requirement Analysis

The system was designed to meet the set of requirements described below to be A System to Detect suspicious Activities in Network Traffic effectively. This requirement analysis is based on the findings from the literature review to come up with requirements. The researcher came up with requirements which were based on their action, ability, and testability.

4.2.1 Functional Requirements

i. The system should be able to allow users to create custom rules on Snort

ii. The system should be able to detect suspicious traffic on the network.

iii. The system should be able to store the logs of the activity in a database that can be accessed by third-party applications (BASE)

iv. The system should allow BASE to interface with Snort to display the network logs on a web interface

4.2.2 Non-Functional requirements

The proposed system was designed to allow the creation and customization of rules and to be able to implement the rules on incoming traffic at all times. Therefore, the system should be able to respond to the incoming packets in a timely manner. The researcher foresees a swift response

time in order to make the users in the network where this system is deployed and tested not experience any further latency added to packet processing time.

The system should be scalable. This means, after implementation, the system can be ported with very minor modifications to any network environment and operation. The fact that the researcher demonstrated the operation of the system on a virtual machine, it is pre-empted that the system would still work the same way when plugged in to a real-world scenario.

4.3 Environment for the System to Detect Suspicious Activities in Network Traffic

In order to come up with this System that Detect Suspicious Activities in Network Traffic, the researcher needed a platform that gives great room for customization of the security tools in use. The system was implemented in a Linux based environment running Ubuntu 16.04 LTS. The choice of the environment was based on the open source nature of the OS and its ability to allow Snort and Base and other supporting components to be integrated with other security tools and features to enhance the operation.

4.4 Snort Analysis and Architecture

Snort as defined by Dietrich (2015), is an open source network intrusion prevention and detection system utilizing a rule driven language, which combines the benefits of signature, protocol and anomaly-based inspection methods. The Open Source tool is hailed for its ability to be able to detect TCP/IP datagrams on a network in real time. In addition, the flexibility of the software makes it very popular among the security researchers. This allows them to connect the software to multiple databases such as Oracle, MySQL. In addition, Snort as Gómez et al. (2009) explain, can be connected with other components of analysis to make the work of analyzing the data easier.

Operations of Snort can be in two ways; it can be configured to work as a packet sniffer like tcpdump and Wireshark tools and as an IDS or IPS. As a packet sniffer, Snort allows the user to view the network activities. However, to allow Snort to act as an IDS, there has to be some rule creation and rule matching operation. Snort comes with some packaged rules that require a little modification to protect against Nmap port scanners, DoS and DDoS attacks as well as backdoor attacks.

The Snort operation is based on the integration of the following components in the architecture diagram shown below

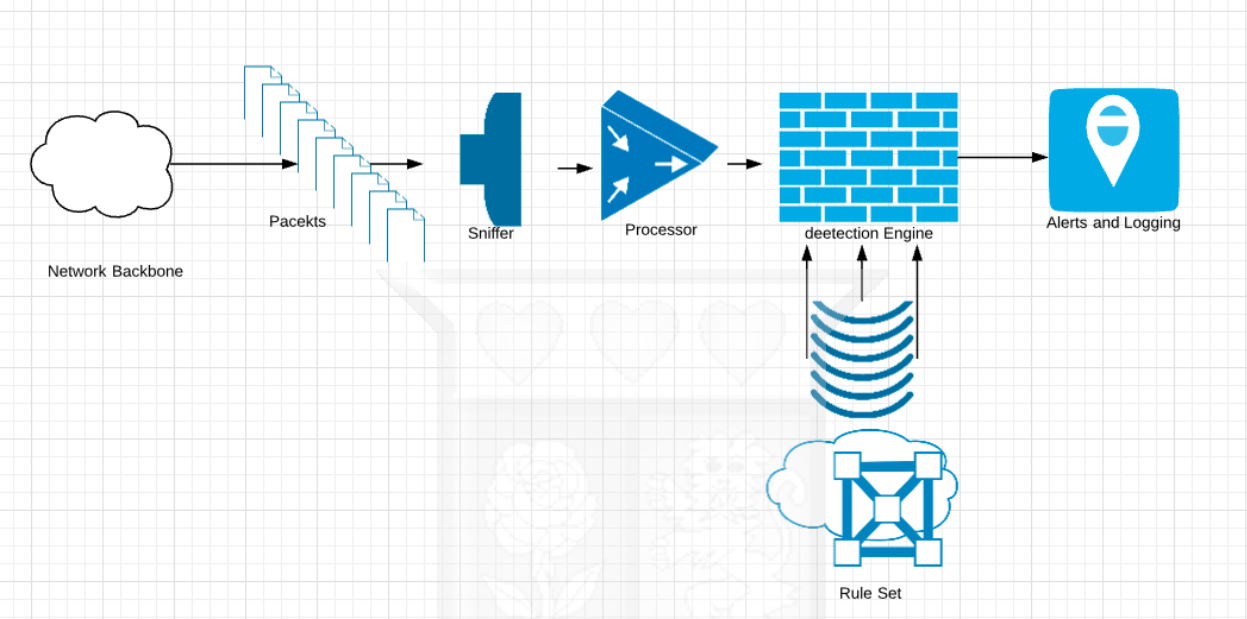


Figure 4-1: Snort architecture diagram (adopted from Anon, 2019)

The sniffer, in the figure 4-3, listens for traffic on the network interfaces and captures the packets. These packets are meant to be decoded and translated in to C language (Gullett, 2011). From there, the pre-processor uses that information to identify the link layer protocols, IP and TCP/UDP in the packets. The pre-processor then filters these packets alongside identifying the properties to be checked by the detection engine. The rule set acts as an input to the detection engine. It populates the detection engine with instructions, which are used to check against the patterns on the packet information that has been obtained from the pre-processor. Finally, the alerting and logging module is responsible for giving feedback to the users and logging the activities of the other components in a database. Some implementation of Snort opts to use log files instead of a database.

4.5 Basic Analysis and Security Engine analysis and architecture

BASE as described by Zhou et al. (2010) does the work of searching through the database that contains the logs collected by network monitoring tools. BASE is open to work with an IDS or a Firewall. The Web GUI written in PHP is used to display and analyze the information from the database in a user-friendly interface. This allows the security researchers or analysts to be able to make inferences and decisions a bit faster based on the filtering capabilities that come with the implementation of BASE as an analysis tool. Its ability to read network layer packets and transport layer packets makes BASE very effective in analysis of network traffic logs. Besides this, the portal is able to generate graphs and statistics based on the customization and queries passed by the user through the engine.

In addition, BASE can be configured as an alerting system. A security researcher can be allowed to classify alerts and manipulate the response of the system to these alerts. Besides that, after major tests, false positives can be detected and deleted. BASE also allows the security administrator to allow the system to send alert messages via email. BASE, in most common implementations, is set to read both tcpdump and Snort log files easily.

4.6 Barnyard2 Overview

According to Erturk, & Kumar (2018) Barnyard2 is an open source interpreter for Snort's unified2 binary output files. It allows Snort to write the logs into the disk in an efficient manner. Through this operation, it makes sure that no network traffic is missed because it does the work of parsing binary data into multiple formats and leaves Snort to only sniff the network for network activities. It can operate in three modes.

In batch mode, it only parses the file specified by the system administrators explicitly and then turn off. While in continuous mode, it starts looking for specific file patterns in a location and then continue to process new data as they appear or as they are exposed to it. Lastly, in continual mode with bookmarking, Barnyard2 uses a *waldo* file, which in Snort means a checkpoint file. The checkpoint file is used to track the progress of the interpretation and in an event where the process ends while the *waldo* file is in use, Barnyard2 resumes processing at the last entry available in the *waldo* file

4.7 PulledPork Overview

This PERL based tool is used alongside Suricata or Snort to perform rule management. It downloads and installs the various rules Snort uses depending on the version of Snort running in the system. Most configurations allow it to run a cronjob to update the rule automatically.

4.8 Diagrammatic Representation of the system

4.8.1 System Architecture

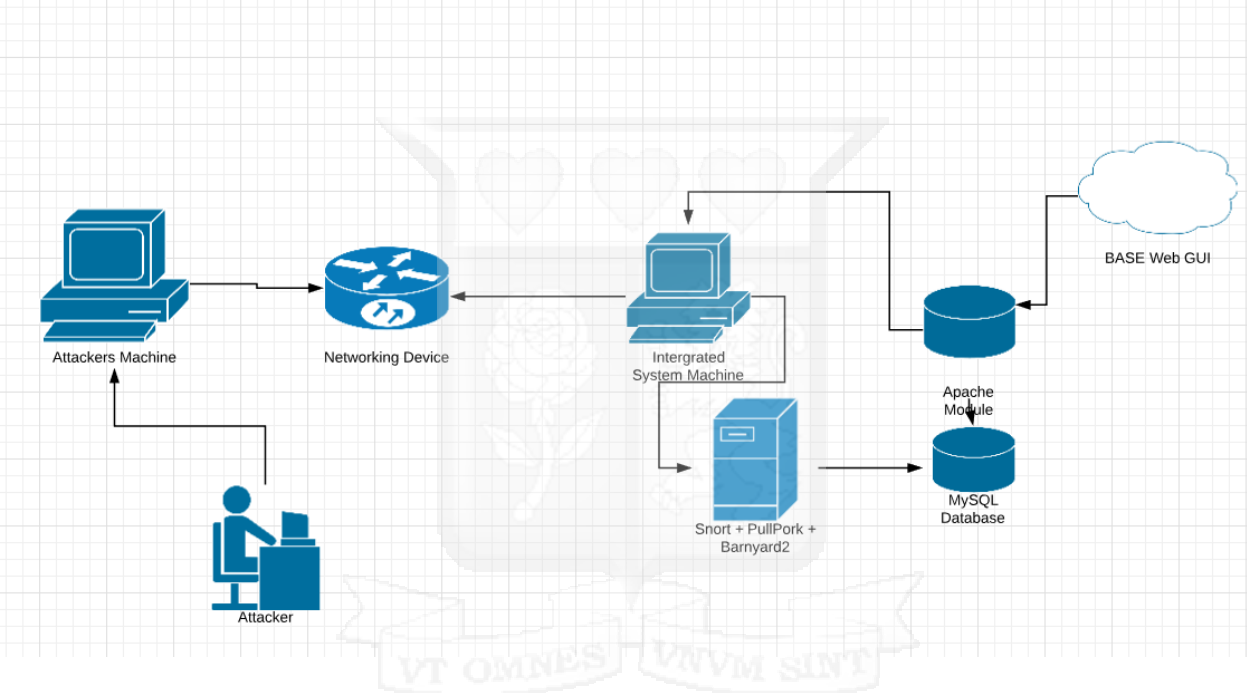


Figure 3-2: System architecture

The figure 4-2 outlines the blueprint of the operation and design of the system. From the outlook, the researcher has presented the attacker who connects to the network. The attacker in this case doubles up as the system administrator and is responsible for running the intergrated security system machine. The intergrated security system machine runs Snort and PulledPork plugin. This machine communicates to Barnyard2 which connects to the MySQL database. The BASE web GUI is hosted in apache on the system's machine which has access to the MySQL database to fetch data.

4.8.2 Context Diagram

The data input and output modules for this research are Snort, the NIDs, Barnyard2, BASE and the User. The user in this case is the researcher who doubles up as the attacker and the system administrator. Snort and Barnyard2 are used to feed data to the Snort DB that the user created. Snort sniffs network traffic and logs raw binary data on HDD. Barnyard2 gathers these data and send to DB. BASE and the user are used as output modules because the user queries from the DB using BASE as the interface and the results are presented.

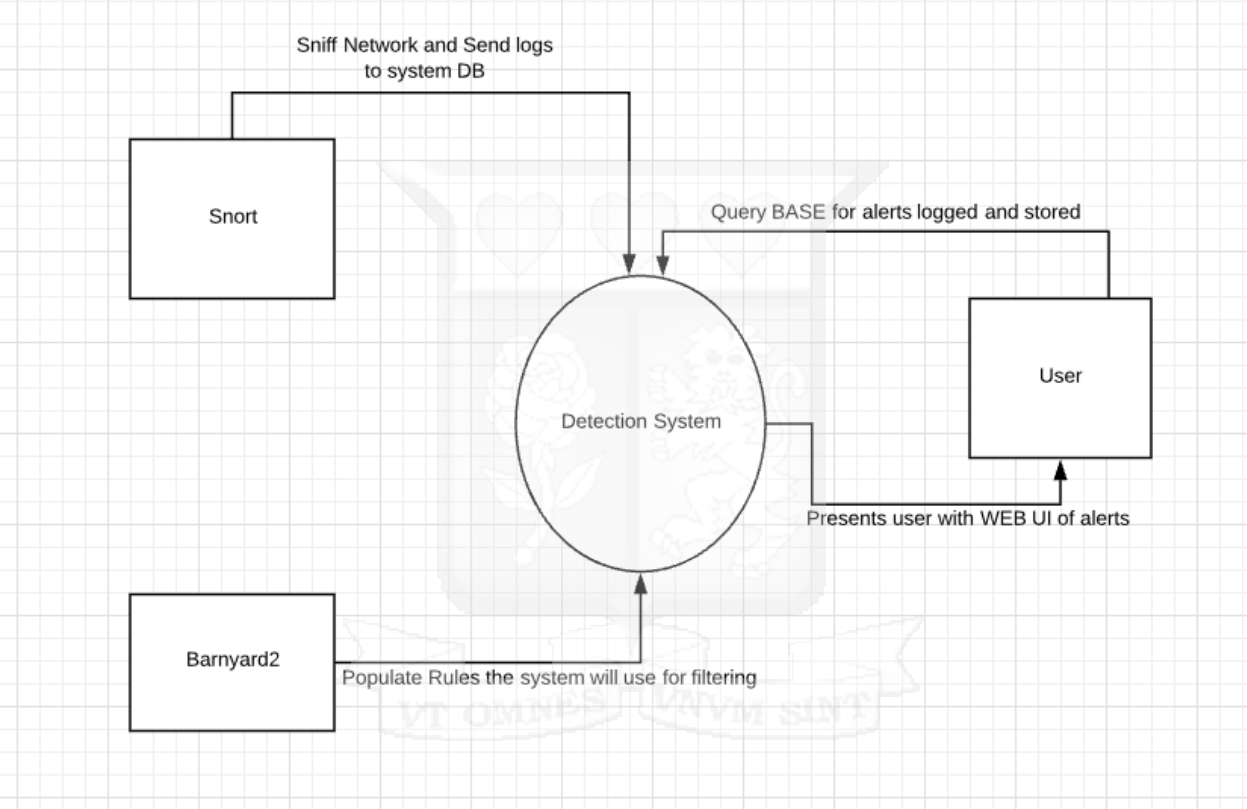


Figure 4-3 : Context Diagram

The figure 4-3 shows the user in this case as the researcher who doubles up as the attacker and the system administrator. Snort and Barnyard2 are used to feed in data to the Snort DB that the user created. Base and the User are used as output modules because the user queries from the DB using BASE as the interface and the results are presented to them.

4.8.3 Use case Diagram

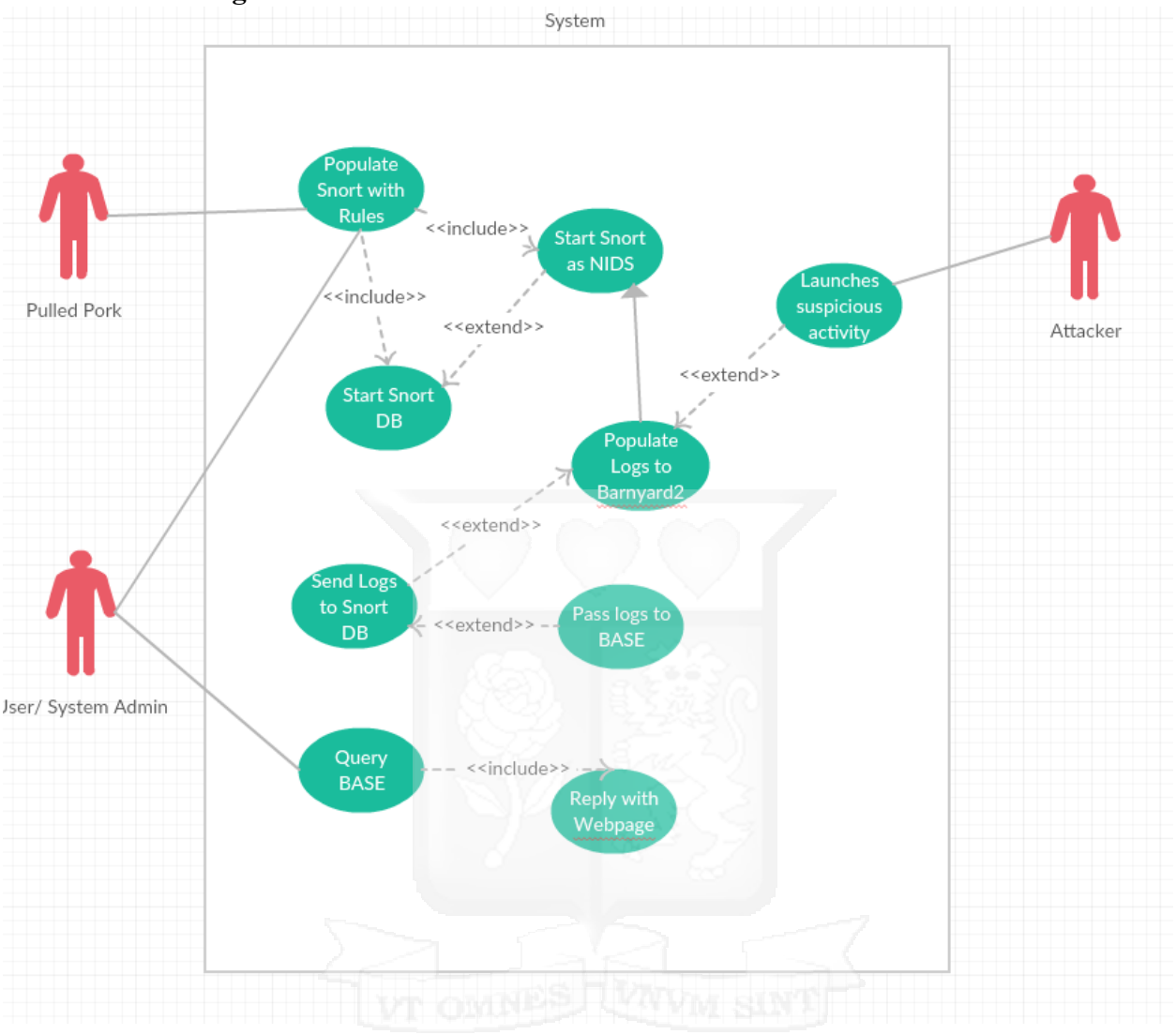


Figure 4-4: Use case Diagram

The figure 4-4 shows how the different actors interact with the system to make the intergrated security system to monitor suspicious network traffic to be implemented. The use case describes three actors. The PulledPork plugin is identifed as an actor because it is to be involved in the population of rules to the system from the snort online community. Population of rules, is also to be done by the User who is to be the researcher in this instance when creating the custom rules.

The population of rules use case invokes two other processes which are starting the Snort DB and initializing Snort to work as a NIDS. The start Snort DB use case is also extended by the populate logs to Barnyard2. Barnyard2 is responsible for sending the logs it collects to the snort

DB. At the same time, population of log files into Barnyard2 is extended when Snort is started as NIDS. Barnyard2 on the other hand extends the sending of logs to Snort DB.

Sending of logs to snort DB means that the database was configured to work with multiple 3rd party applications. This procedure invokes and extends to passing of the logs to BASE. BASE is the third party application that the researcher settled for to collect the logs files that have been stored in Binary format in the snort DB.

Finally, the user from the usecase diagram is to be responsible to query the BASE engine for results on the web interface. The Base engine responds to the users queries by presenting them with a web page containing their logs captured by Snort in the Snort DB.

4.8.4 System sequence Diagram

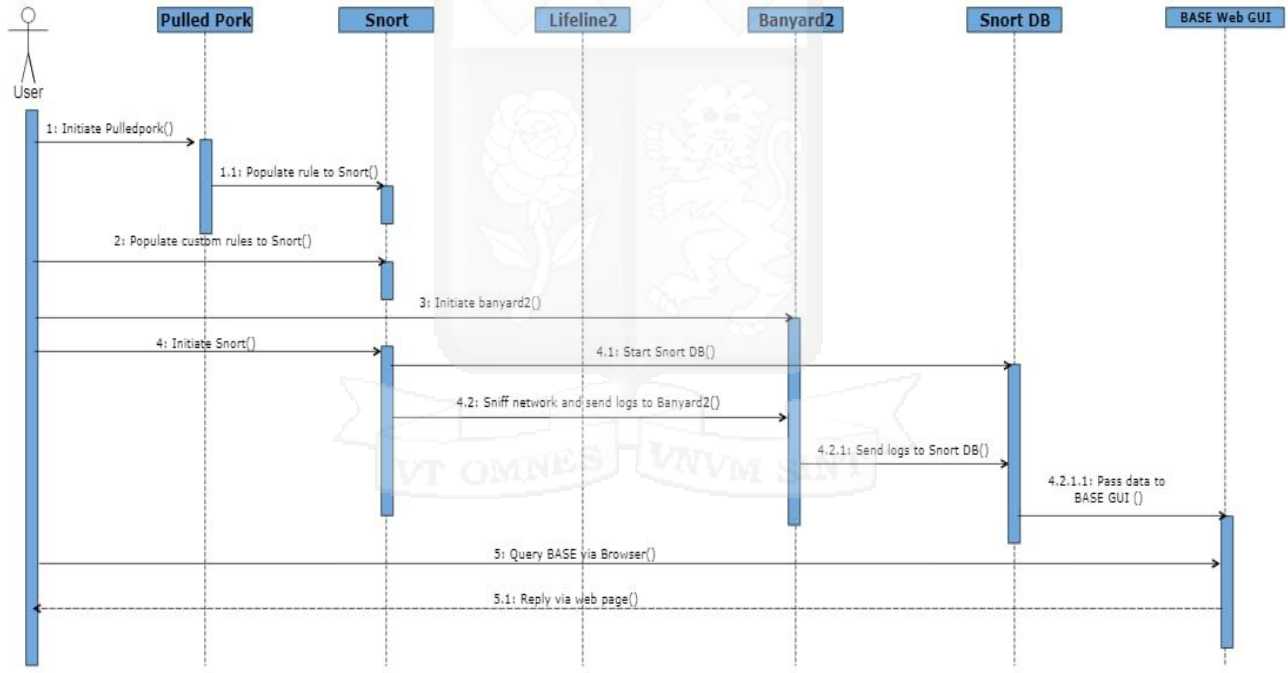


Figure 4-5: System sequence diagram

The sequence diagram in figure 4-5 illustrates the flow of events when implementing and testing the integrated security system to monitor suspicious activities on a network. The first event is the initiation of PulledPork by the user to ensure that rules from Snort's website are fetched and populated on snort's configuration files.

The next activity is the user writing custom rules in Snort's configuration folders. After this, the user invokes Barnyard2 to listen on what Snort is capturing. After this, the user initiates Snort to start and work as a NIDS, at this point Snort and Barnyard2 interface. The initiation of snort also starts a sub process in the background that is the Snort DB that is be used to collect data sent to it by Barnyard2.

Snort scans the network and send the logs to Barnyard2. Barnyard2 parses and changes the logs into binary format and sends them to Snort DB for storage. Since this work allows BASE to access the snort DB, the logs from Snort DB are made accessible to BASE. The user then initiates a query to BASE asking it to display the web page. BASE interprets the user's request and replies with a web page.

Chapter 5

Implementation and Testing

5.1 Introduction

This chapter builds up on the design and analysis from the previous chapter. The outline of the chapter is as follows; the models that were brought together to build the system are discussed. Both hardware and software components are discussed. The chapter then describes the setup of the experiment. It outlines all the steps that were involved to setup the environment. The chapter goes through a walkthrough of the implementation phase. Finally, the system is tested and the test results are presented and analyzed.

5.2 Hardware, Software and Model Components

The software and hardware components that were used for the experiment to be realized in this research were informed by the nature of the tests to be conducted to achieve the objectives set out by the research.

1. Hardware components, A Personal Computer
    i. 8 Gigabyte (GB) Random Access Memory (RAM)
    ii. 500GB hard drive disk storage
    iii. Intel core i5-7200u OS at 2.7 GHz of processing speed
2. Software Components
    i. Windows 10 Host Operating System
    ii. Virtual Box Hypervisor
    iii. Ubuntu Server 16.04 image (Guest OS)- Running Snort and BASE
    iv. Windows 10 operating system - Attacker's machine
    v. Putty terminal emulator, serial console and network file transfer application.
3. Model components

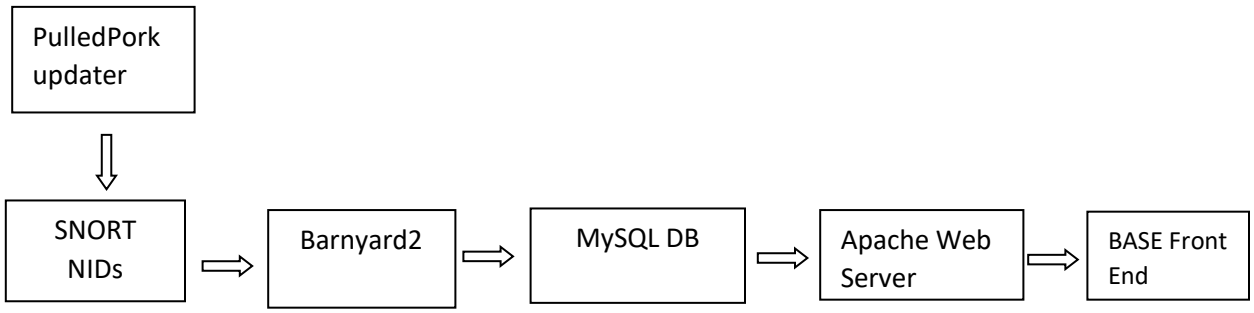


Figure 5-1: Model diagram

5.3 Experiment Setup

The experiment to develop the System to Detect Suspicious Activities in Network Traffic was done on an Ubuntu 16.04 machine that was customized to meet the needs and objectives of the research. Snort, which was the main software component to sniff packets in the network was installed alongside other supporting components. It was very important for the researcher to make sure Snort is downloaded and configured as a NIDS. Other components that the researcher opted to install to support Snort included Barnyard2 module, PulledPork and BASE.

5.3.1 The Steps followed to set up the experiment

To get started virtual box and its extension pack were downloaded and installed on Windows to create a virtual machine. The next step was to install and customize Ubuntu 16.04 LTS operating system. Large Receiving Offload and Generic Receiving Offload that do packet reassembly were disabled to enable snort to truncate packets that were larger than 1518 bytes. Setting up snort on Ubuntu 16.04 LTS consisted of downloading the file, installing and configuring it to run as NIDs. Snort was configured to run as NIDs by editing some configuration files. Testing of snort was done by using another terminal (Test mode). This was important so as to maintain the status quo and be able to move to the next step of integrating Barnyard2. Barnyard2 and its dependencies were downloaded and installed. Barnyard2 was then configured for use with MYSQL and as a result Snort DB was created. The researcher went ahead and tested Snort by running it in alert mode using the config file. This process ensured Barnyard2 and snort were integrated successfully. PulledPork and its dependencies were then set up. Testing of PulledPork was important so as to confirm the configuration changes had been affected. Cron was then downloaded and installed for purposes of scheduling PullePork. BASE provides a web front end to query and analyse the alerts coming from the NIDs system. Hence the need to download and install it together with its dependencies. Testing was done by launching a local browser to confirm whether it was working. Http://localhost/base. Then went ahead and configured BASE. Lastly, start-up scripts were created for Snort and Barnyard2 daemon.

5.4 System Implementation

5.4.1 Setting up the network interfaces and installing Snort

The Large Receiving Offload (LRO) and the Generic Receive Offload (GRO) that do the packet reassembly as captured by the network interface Snort is running on can affect the network interfaces running Snort. Therefore, they need to be disabled during setup to enable Snort to truncate packets that are larger than 1518 bytes.

This was done by editing the network interfaces file in the Ubuntu 16.04 files and adding the following two lines ". *post-up ethtool -K enp0s3 gro off, post-up ethtool -K enp0s3 lro off*. "in the specific interface file under the /etc/network/ directory. The results were confirmed as shown in Appendix A.

The Snort installtion was to take place in a new directory in the desktop of the ubuntu system. The tarball file was downloaded from Snort's website, extracted and compiled. The libraries were later updated. More information of the commands that were run throughout the installation process are captured in the Appendix H. However, the resulting screen to demonstrate the correct installation of Snort is in Appendix B.

5.4.2 Configuring Snort to work as an IDS

Installation of Snort was not enough to allow it to monitor and scan the network activities. Steps that the resercher took to achieve this included copying and setting up the configuration files of Snort folder from the desktop where the tarball was downloaded to the /etc directiory in the Ubuntu system. Besides that, the researcher also set up the log collection point of Snort to the /var/log folder for easier management.

In addition, the researcher had to configure the main configuration file that runs Snort and add the network that Snort should sniff and collect data from. Since this implemention was being done in different environments, work, home and campus with different internet subnets, the IP address subnet indicated varied depending on the location of the researcher. The configuration file image results is shown below.

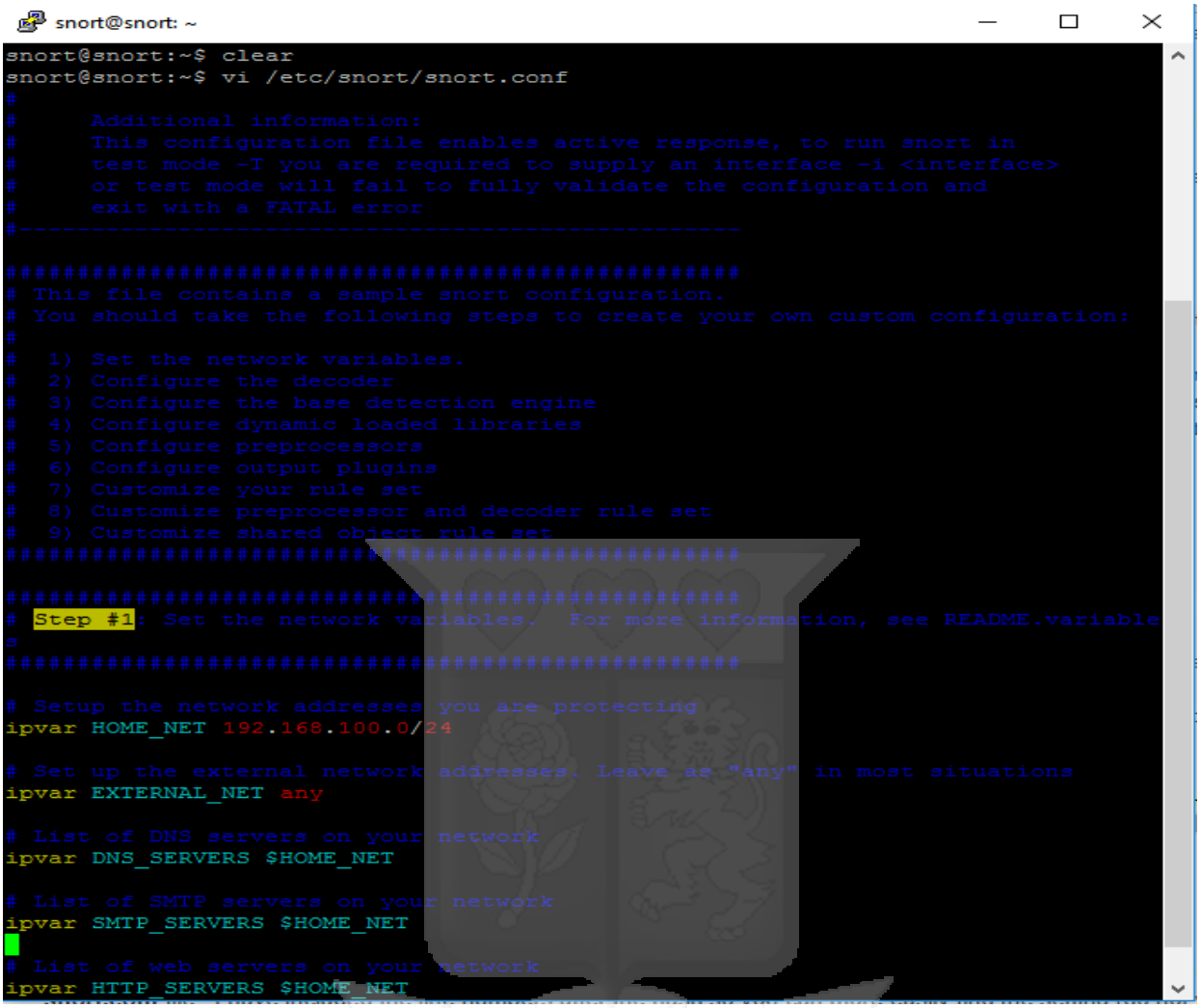


Figure 5-2: Snort.conf configuration details

A series of the commands that the reseacher executed to achieve this is attached on the appendix I.

5.4.3 Barnyard2 Installation and setup

The installation of Barnyard2 required installation of dependencies which included downloading and setting up a MySQL DBMS that was used to store the log files that were captured by Snort from the analysis of the network traffic. The DBMS had a database the researcher called Snort. The researcher then had to download Barnyard2 from the source and make changes to point it to the Snort Database and allow it to store files there. This was achieved by configuration of the main config file that Barnyard2 runs on and adding the output database configuration line at the end. The images below show the configuration of MySQL database, linking the database with

Barnyard2 and correct initialization of Barnyard2. Successful installation of Barnyard2 and linking with MySQL is shown in Appendices C to E

5.4.4 PulledPork Installation and Setup

To set up PulledPork , the dependencies and the main packages had to be downloaded. The tarball file that PulledPork comes with contains config files that needed to be copied to the /etc folder. The pulledpork.conf file, which is the main config file had to be edited to allow the application to pull the rules to the correct destination in the Snort folders. A summary of the lines to be changed is shown below.

```
Line 19:  enter your oinkcode where appropriate (or comment out if no oinkcode)
Line 29:  Un-comment for Emerging threats ruleset (not tested with this guide)

Line 74:  change to: rule_path=/etc/snort/rules/snort.rules
Line 89:  change to: local_rules=/etc/snort/rules/local.rules
Line 92:  change to: sid_msg=/etc/snort/sid-msg.map
Line 96:  change to: sid_msg_version=2

Line 119:  change to: config_path=/etc/snort/snort.conf

Line 133:  change to: distro=Ubuntu-12-04

Line 141:  change to: black_list=/etc/snort/rules/iplists/black_list.rules
Line 150:  change to: IPRVersion=/etc/snort/rules/iplists
```

Figure 5-3: PulledPork.conf configuration summary

5.4.5 Installation of BASE and Set Up

To install BASE, the reseacher had to lay the foundation. Apache2 web server had to be installed and configured correctly to run PHP plus other dependencies. Adodb, which is a database abstraction layer for PHP had to be installed in the system. This application allows a wide range of codes to be used to access a database in the background. The next procedure was to download BASE and copy the files in Apache root folder /var/www/html to allow it to be accessed from the browser. The main configuration file that runs BASE had to be edited to allow it to connect to Adodb and the Snort database the application was running in MySQL. A summary of the lines that were edited in the /var/www/html/base/base_conf.php files is shown below.

```
$BASE_urlpath = '/base';                    # line 50
$DBlib_path = '/var/adodb/';                #line 80
$alert_dbname     = 'snort';                # line 102
$alert_host       = 'localhost';
$alert_port       = '';
$alert_user       = 'snort';
$alert_password   = 'MySqlSNORTpassword';   # line 106
```

Figure 5-4: base_conf.php configuration summary

After the correct setup of BASE. The, web console needed to be accessed to confirm the operation of the BASE module. The path was as follows :192.168.100.14/base

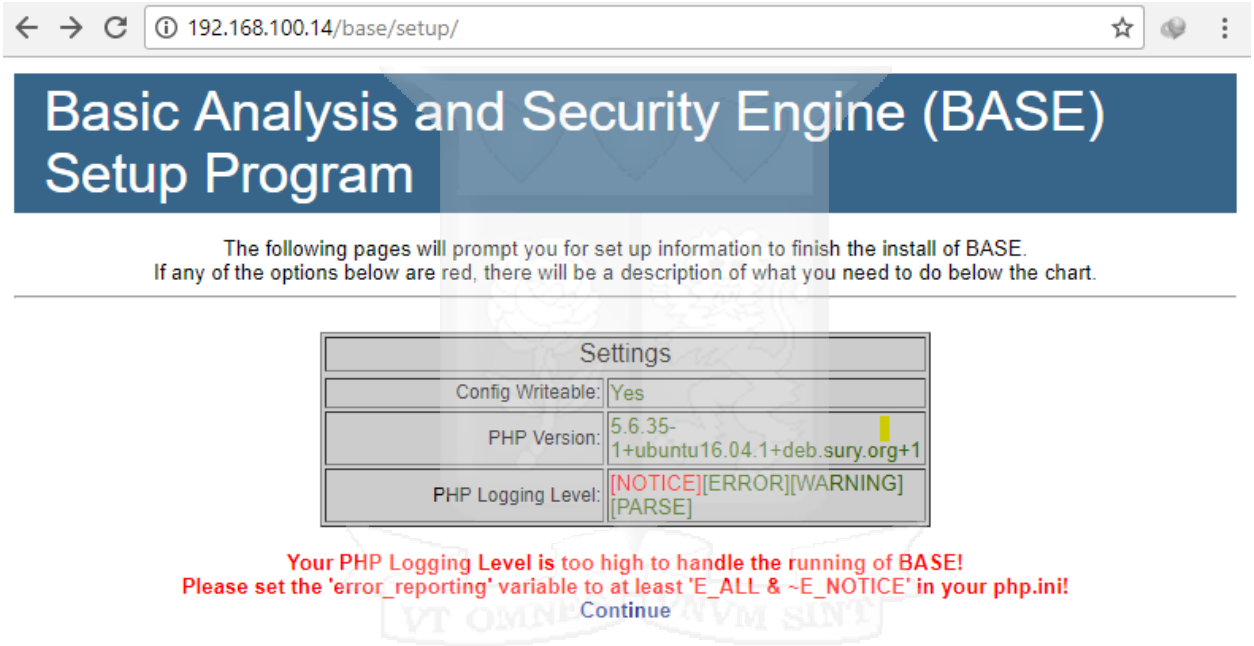


Figure 5-5: Successful BASE setup and configuration

5.5 System Testing

The experiments that were conducted by the research in order to come up with A System security tool that allows security administrators to run network scans was done by the invocation of two custom rules that were written on Snort. The rules that were written were supposed to capture erroneous ICMP requests being sent in the network specified in Snorts range and sending that data through the spooler, Barnyard2 to the MySQL database and finally allowing BASE to pull the results on a web interface.

5.5.1 Custom rules definition

The researcher needed to write down custom rules to the system to allow the system to be able to detect the ICMP packets sent to it from the network and send those logs via the spooler (Barnyard2) for storage. The illustration of the two rules written is captured by the image shown below.

Figure 5-6: Local rules added to Snort configurations

The variable $HOME_NET was prefered by the researcher because of the changing nature of the network addresses whenever the researcher was moving in different networks when configuring the system.

5.5.2 A sample ICMP larger than 64 bytes requests

To test the operation of the system, the researcher posed as the attacker on their host machine and launched an ICMP DoS kind of attack by sending 100 large ICMP packets to the network and Snort was able to detect.

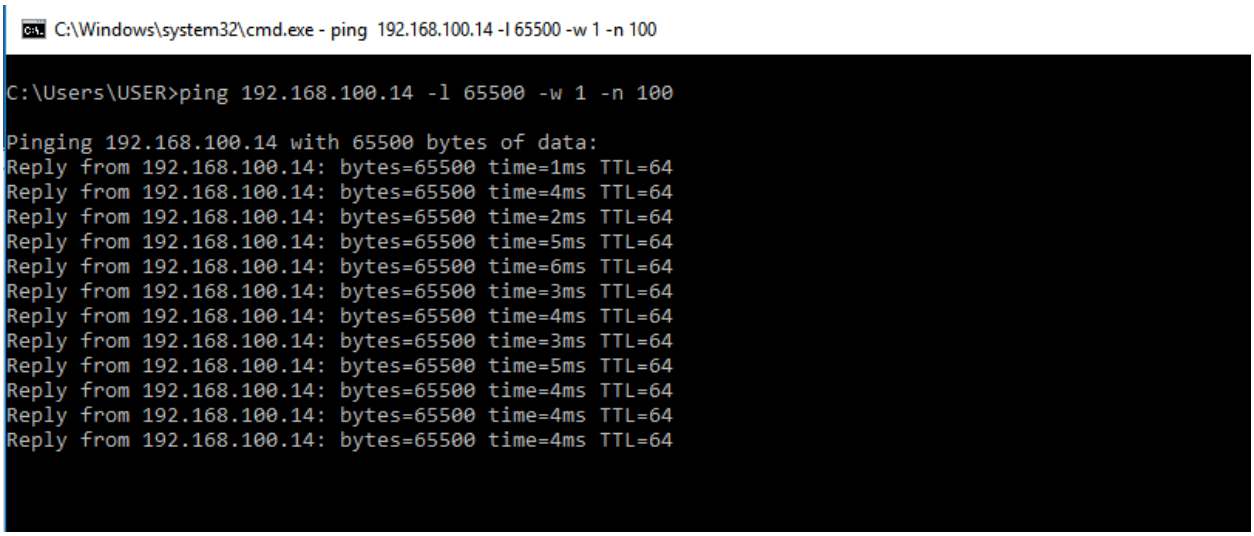


Figure 5-7: Sample DoS attack launched with ICMP

5.5.3 Snort Console results to demonstrate Banyard2 and Snort Sniffing Capabilities

The image below demonstrates how Snort was able to capture the rouge ICMP requests sent to it by the user at IP address 192.168.100.9. The captured packets were then passed and sent to the MySQL database by Barnyard2 for storage and collection by BASE

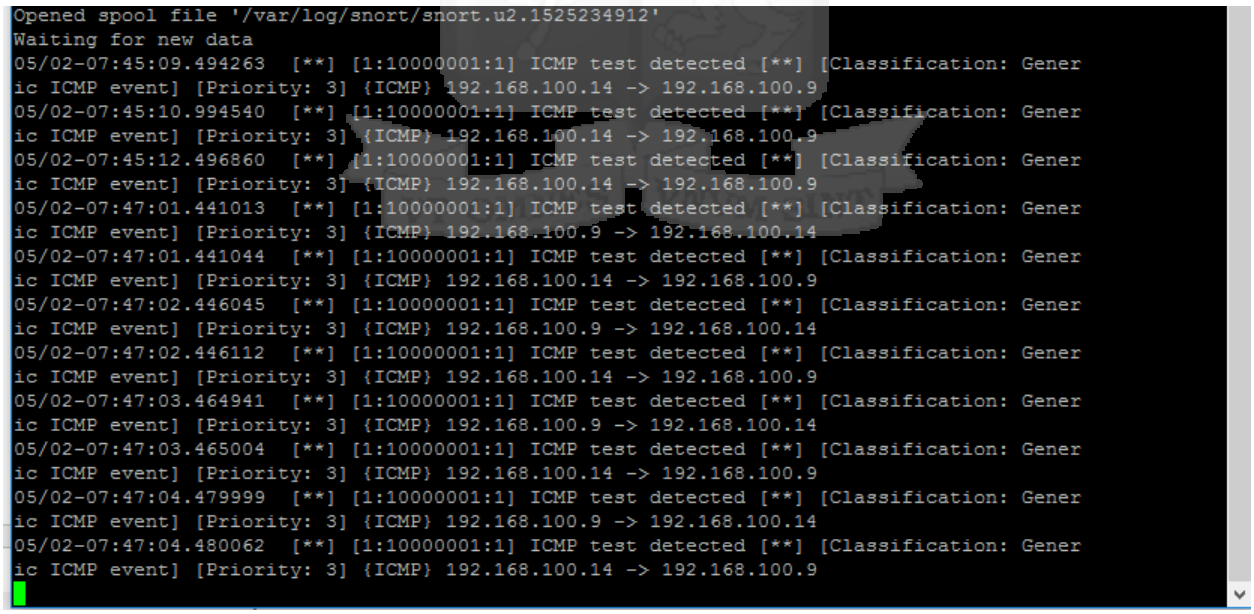


Figure 5-8: Snort detecting ICMP request sent to it

5.5.4 BASE portal to showcase the results from Snort

The ICMP requests that were sent to the network, captured by Snort and stored in the database, were able to be captured by BASE and displayed on the web GUI as expected. The images captured below show the results from BASE Web GUI for the ICMP attacks launched to the network. Detailed screens that depict the results from Snort are captured in Appendices F and G

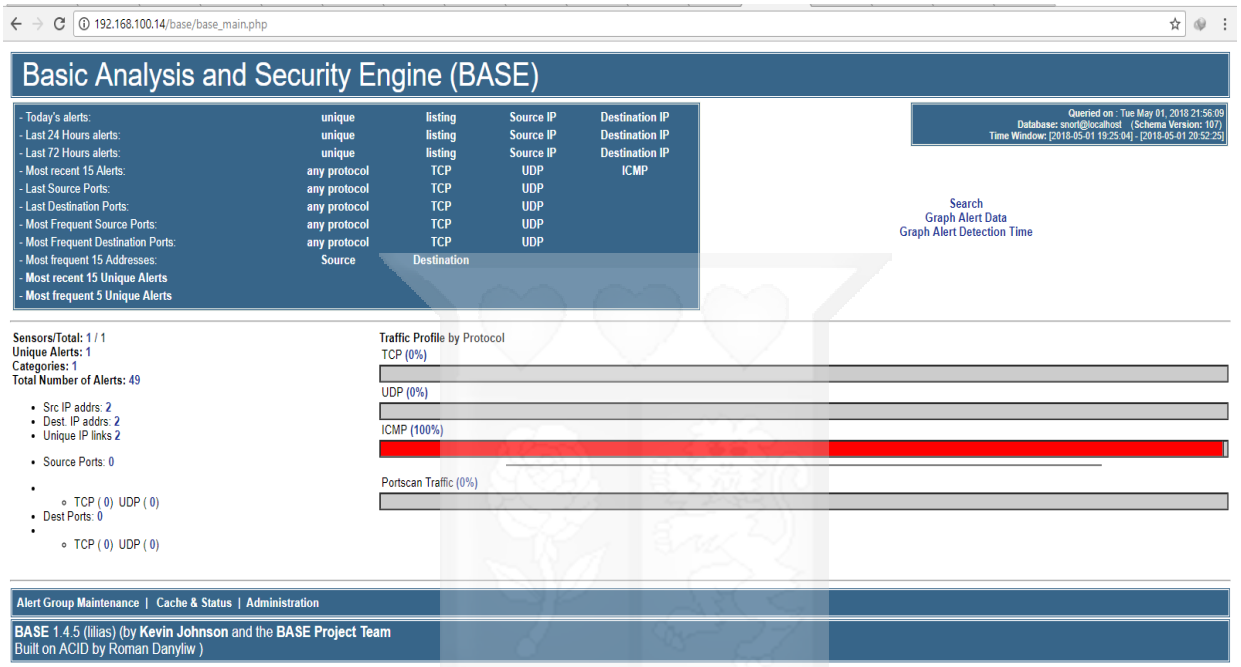


Figure 5-9: BASE homepage with summarized dashboard

5.5.5 System Testing Classes

Table 1. System testing classes

Test Class	Inspection Check	Priority
Functional	Were there any custom rules created? In addition, did the rules take effect in the Snort system to allow it detect traffic based on the customized rules?	High
Functional	Was the system able to detect suspicious traffic on the network	High

Functional	Was the system able to store the logs of the activity in a database MySQL that can be accessed by third-party applications (BASE) after referencing to it?	Medium
Functional	Did the system allow BASE to interface with Snort to display the network logs on a web interface?	High
Non- functional	Was the system responsive enough to respond to the incoming packets in a timely manner and present results to the system administrator in a real time fashion?	Medium

5.5.6 System testing results

Table 2. System testing results

Test Class	Test Results	Comment
Functional	Pass	Suspicious ICMP traffic was able to be detected in the network by the integrated security system and the results presented in an easy to use GUI powered by BASE
Non-Functional	Pass	There was acceptable responsiveness from the system during the implementation as it was able to capture real time events in the network traffic.

5.6 Challenges faced

The implementation and testing of the system presented the researcher with a number of challenges. The complexity of customizing the Ubuntu 16.04 and making sure the components that allowed the implementation of a system security system was faced. Allowing the modules, Snort, PulledPork, Barnyard2 and BASE to work in harmony required a lot of tenacity and

research to understand their different functionality. Besides that, another complexity that the researcher was faced with was the configuration of the dependency packages. The PHP module selected by the researcher at first was not compatible with BASE and PERL. The researcher had to look for a solution, which was to downgrade to a lower version.

The other challenge that the research posed during the implementation was constantly changing the Snort configuration file to capture the new network configuration that was presented by the new network the researchers' machine visited. This meant that the researcher was always aware of the changes and any negligence whenever they moved to a new network would make the system not function properly.

Chapter 6

Conclusions and Future Work

6.1 Overview

This chapter dissects the finding presented in the chapter above. It discusses the experiment results and sheds light between the topic of the research and the findings presented above. In addition, the chapter looks at the future research work that could be done in the same area by concluding and making recommendations drawn from the research findings.

6.2 Discussion

Data collected in the previous chapter brings into perspective why integration in security operations (threat detection, alert system, analysis and response) is very important. From the test results, the researcher was able to pose as the attacker in the network and launch an ICMP attack similar to a DoS attack. The system was able to catch this suspicious activity in the network and respond to it appropriately. This was due to the functionality of Snort and PulledPork, the network sniffer that was configured to work as a NIDS.

Snort was able to work alone at first by displaying the log files on the terminal. However, top push the agenda for integration, the researcher was able to come up with additional components and plug them in on the system. Barnyard2 was able to pick these snort logs, translate them into binary and push them to the database for storage.

BASE on the other hand came in to provide a good GUI on web platform to allow the researcher to display the alerts in a more organized approach. The different images that show the BASE GUI captured by the researcher in the previous chapter focus on the ICMP attack that was carried out in the network. The screen demonstrates the power of BASE in analysis of those log as it gives more information or very specific and useful information that can be used for decision making as far as security is concerned. For instance, the IP addresses, both source and destination are displayed for all the alerts generated by Snort. This is coupled with the layer 4 protocol information.

6.3 Conclusion

The system was developed primarily to look at the network activities that occurred within the researcher's subnet. Combining the different components that have been discussed in chapter four of the research, gave the work an upper hand in the integration of a security system that would achieve the objectives set out in chapter 1.

Chapter 2 of this research presented the different tools that have been used to implement integrated security systems either for the network or the hosts, which are popularly known as HIDS. The approach that was used to make sure the components the researcher settled for in this work was based on this previous implementation like Elastic Search, Logstash and Kibana implementations. These were the findings that informed the researcher to settle on the research approach that they took in chapter 3 and perform similar software modelling techniques that were employed by the previous works in the field of integrating software components to either NID or HID.

6.4 Recommendations

This research recommends that for a similar setup, security engineers should implement their systems on a Linux based environment. This is because of the hardened nature of Linux based operating system. This means that the researchers were not be worried about the security of their own testbed but the network.

Lastly, the paper recommends that the Snort rules that come pre-packaged by the application is a very good starting point for out of the box implementation. However, it is advised that the security administrators go out of their way to come up with new rules that locally apply to their integration and are custom - made to them. This makes it easier for them to know what each alert means and the reason the alert was triggered. Besides that, it is also important for the security administrators to familiarize themselves with existing snort rules and know what they mean before invoking them.

6.5 Future Research Work

This work focused on installing the system on a network that is running on one subnet. In future, the researcher recommends allowing the system to scan more than one network. In addition, the researcher recommends that for future implementations, important network nodes like DNS,

DHCP, Email and Active Directories should be listed down in the snort.conf file and allow PulledPork to specify specific rules from the open source community to be applied specifically to these important services as the first priority. This means, each of the services will have its own specific rules attached to it and enforced to run on the highest priority.

In addition, this research notes that Snort was configured to run on one instance with base. The research recommends that future works that go into this field should look at how there can be multiple instances of snort in different LANs, owned by the same organization communicating (sending and parsing logs) to one database that BASE will access. The research foresees that this implementation will push integration and the benefits that comes with this centralization a notch higher.

References

Ahn, G., Kim, K. Y., & Jang, J. S. (2011). *U.S. Patent No. 7,882,556*. Washington, DC: U.S. Patent and Trademark Office.

Andersson, M. (2017). A study of Centralized Network Intrusion Detection System using low end single board computers.

Ayyagari, A., Aldrich, T. M., Corman, D. E., Gutt, G. M., & Whelan, D. A. (2015). *U.S. Patent No. 9,215,244*. Washington, DC: U.S. Patent and Trademark Office.

Astels, D. (2003). *Test driven development: A practical guide*. Prentice Hall Professional Technical Reference.

Beck, K. (2003). *Test-driven development: by example*. Addison-Wesley Professional.

Bhuyan, M. H., Bhattacharyya, D. K., & Kalita, J. K. (2014). Network anomaly detection: methods, systems and tools. *IEEE communications surveys & tutorials*, *16*(1), 303-336.

Dietrich, N. (2015). Snort 2.9. 7. x on Ubuntu 12 and 14. *Software User Manual), the SNORT team*.

Dietrich, N. (2015). Snort 2.9. 8. x on Ubuntu 12, 14, and 15 with Barnyard2, PulledPork, and Snorby.

Elastic Blog. (2018). *Analyzing network packets with Wireshark, Elasticsearch, and Kibana*. [online] Available at: https://www.elastic.co/blog/analyzing-network-packets-with-wireshark-elasticsearch-and-kibana [Accessed 27 Jan. 2018].

Ertoz, L., Eilertson, E., Lazarevic, A., Tan, P. N., Kumar, V., Srivastava, J., & Dokas, P. (2005). Minnesota Intrusion Detection System.

Erturk, E., & Kumar, M. (2018). New Use Cases for Snort: Cloud and Mobile Environments. *arXiv preprint arXiv:1802.02359*.

Gómez, J., Gil, C., Padilla, N., Baños, R., & Jiménez, C. (2009, June). Design of a snort-based hybrid intrusion detection system. In *International Work-Conference on Artificial Neural Networks* (pp. 515-522). Springer, Berlin, Heidelberg.

Gullett, D. (2011). Snort 2.9. 0.5 and Snort Report 1.3. 1 on Ubuntu 10.04 LTS Installation Guide. *Symmetrix Technologies*.

Gupta, Y. (2015). *Kibana Essentials*. Packt Publishing Ltd.

J. Cilluffo, F., Madon, M., & Bejtlich, R. (2015). *GLOBAL PERSPECTIVE ON CYBER THREATS*. Washington DC. Retrieved from https://www.hsdl.org/?view&did=790874

Kuc, R., Rogozinski, M., In Hussain, A., In Youe, R., In Srivastava, S., & In Siddiqui, S. (2015). *Mastering elasticsearch: Further your knowledge of the elasticsearch server by learning more about its internals, querying, and data handling*.

NEEDHAM, A. B. R., & Lampson, B. (2008). Network Attack and Defense. *white paper*.

Oktay, U., & Sahingoz, O. K. (2013, May). Proxy network intrusion detection system for cloud computing. In *Technological Advances in Electrical, Electronics and Computer Engineering (TAEECE), 2013 International Conference on* (pp. 98-104). IEEE.

Owuor, H. (2016). *Apache-Based Sever Security Event Monitoring Logging and Alerting System*. Undergraduate. Strathmore University.

Rehman, R. U. (2003). *Intrusion detection systems with Snort: advanced IDS techniques using Snort, Apache, MySQL, PHP, and ACID*. Prentice Hall Professional.

Specht, S. M., & Lee, R. B. (2004, September). Distributed Denial of Service: Taxonomies of Attacks, Tools, and Countermeasures. In *ISCA PDCS* (pp. 543-550).\

Stallings, W., Brown, L., Bauer, M. D., & Bhattacharjee, A. K. (2012). *Computer security: principles and practice* (pp. 978-0). Pearson Education.

Turnbull, J. (2014). *The book*.

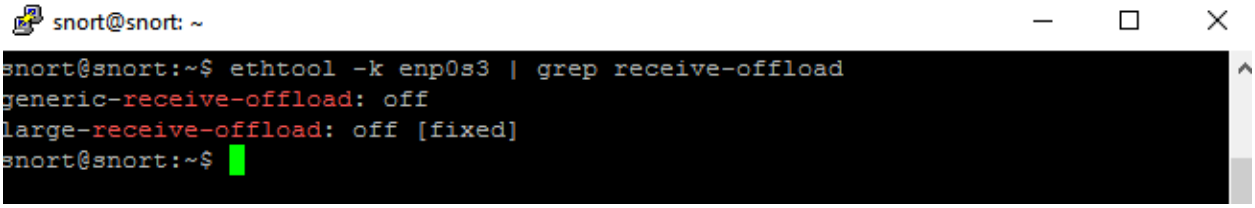
Vidya, S., Gowri, N., & Bhaskaran, R. ARP traffic and Network Vulnerability. *proceedings of INDIACOM-2011, conducted by BVICAM, New Delhi, India, page–619 and in CD*.

Visbal, A. (2015). *U.S. Patent No. 9,100,428*. Washington, DC: U.S. Patent and Trademark Office.

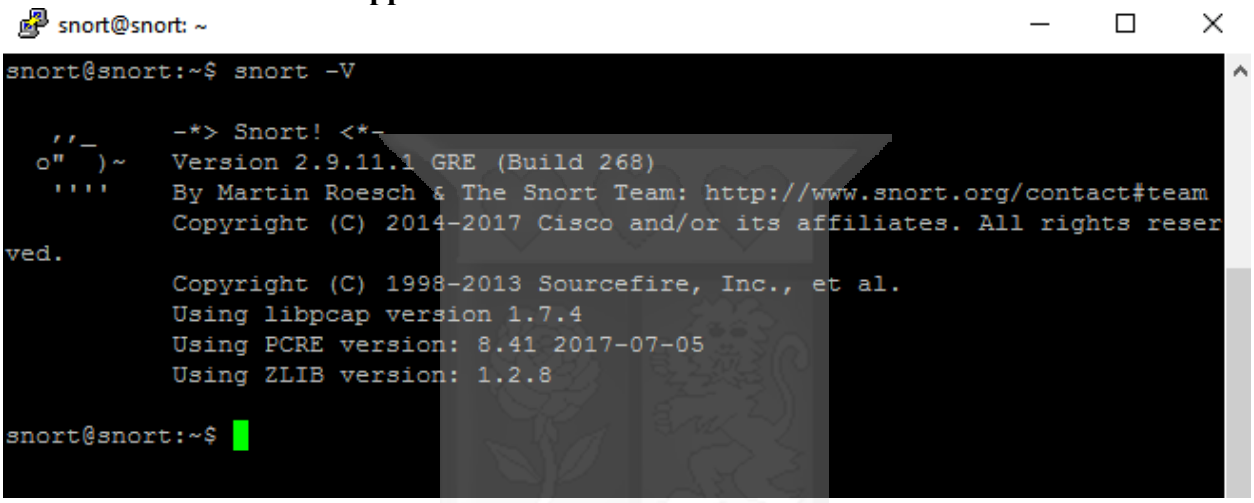
Zhou, Z., Zhongwen, C., Tiecheng, Z., & Xiaohui, G. (2010, May). The study on network intrusion detection system of Snort. In *Networking and Digital Society (ICNDS), 2010 2nd International Conference on* (Vol. 2, pp. 194-196). IEEE.

Appendices

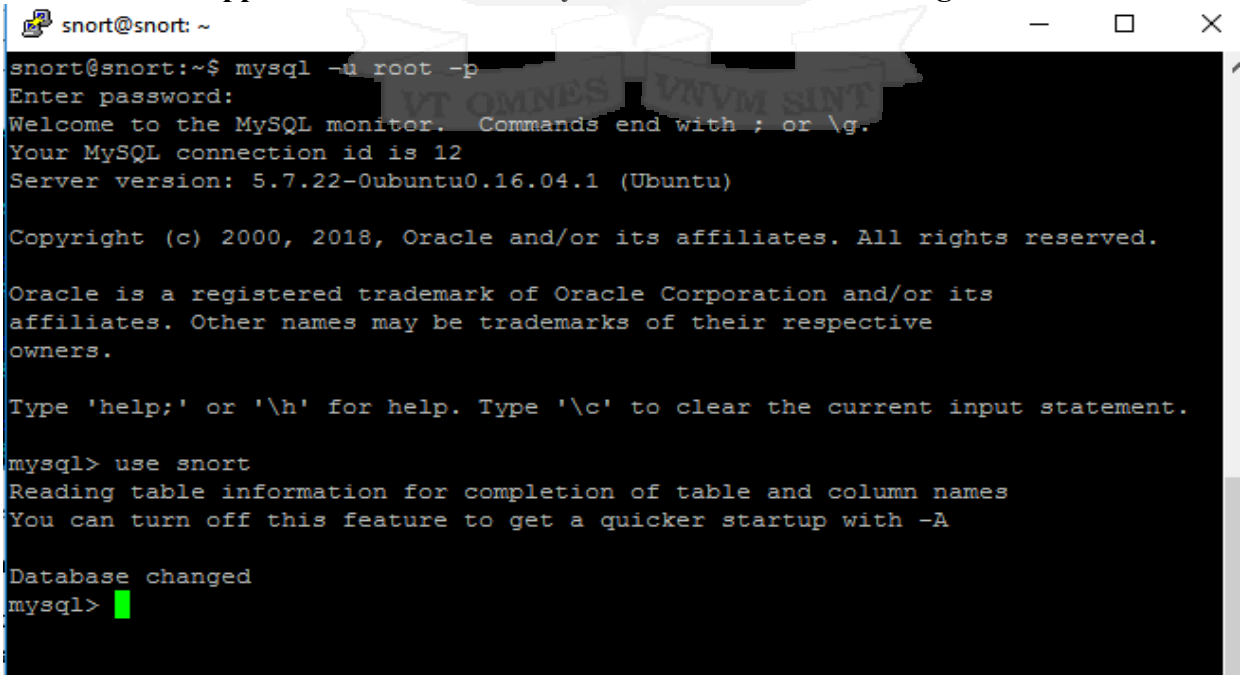
Appendix A: LRO and GRO disabled feedback



Appendix B: Successful snort installation



Appendix C: Successful MySQL installation and configuration



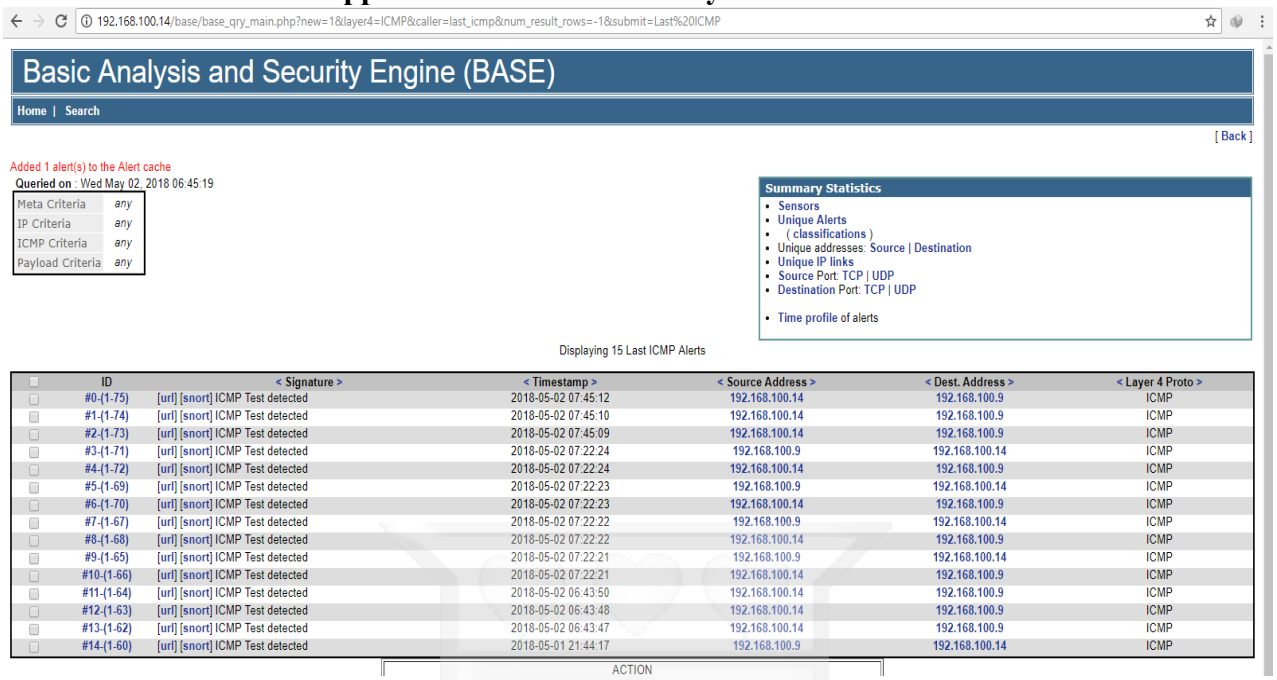
Appendix D: Successful Barnyard2 integration with MySQL

```
# Examples:
#
# output alert_fwsam: snortsambox/idspassword
# output alert_fwsam: fw1.domain.tld:898/mykey
# output alert_fwsam: 192.168.0.1/borderfw  192.168.1.254/wanfw
#
output database: log, mysql, user=snort password=snort dbname=snort host=localhost sensor
 name=sensor01
~
                                                              375,0-1       Bot
```

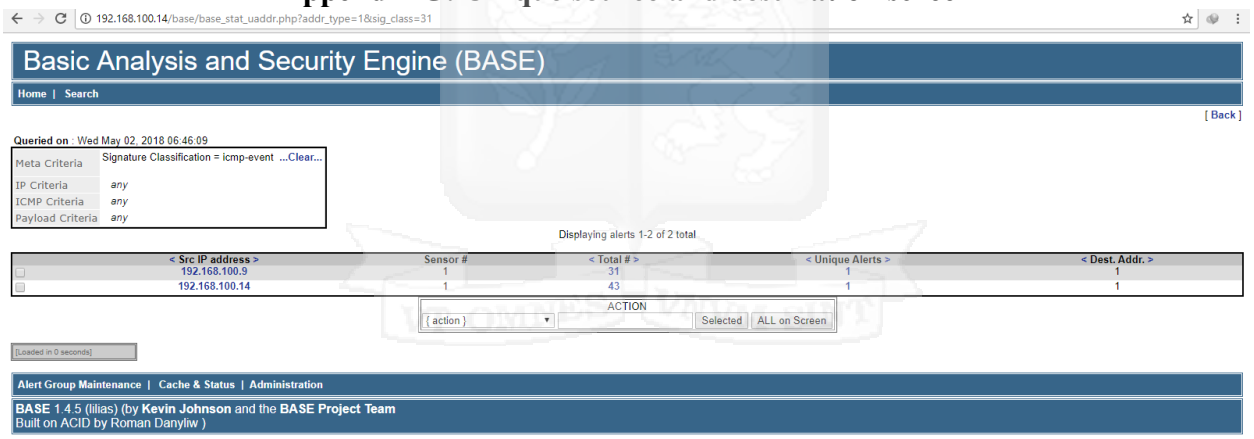
Appendix E: Successful Barnyard installation and configuration

```
snort@snort: ~
snort@snort:~$ sudo barnyard2 -c /etc/snort/barnyard2.conf -d /var/log/snort -f snort.u2
-w /var/log/snort/barnyard2.waldo -g snort -u snort
Running in Continuous mode

        --== Initializing Barnyard2 ==--
Initializing Input Plugins!
Initializing Output Plugins!
Parsing config file "/etc/snort/barnyard2.conf"


+[ Signature Suppress list ]+
----------------------------
+[No entry in Signature Suppress List]+
----------------------------
+[ Signature Suppress list ]+

Barnyard2 spooler: Event cache size set to [2048]
Log directory = /var/log/barnyard2
INFO database: Defaulting Reconnect/Transaction Error limit to 10
INFO database: Defaulting Reconnect sleep time to 5 second
database: compiled support for (mysql)
database: configured to use mysql
database: schema version = 107
database:           host = localhost
database:           user = snort
database:  database name = snort
database:    sensor name = snort:NULL
database:      sensor id = 1
database:     sensor cid = 73
database:  data encoding = hex
database:   detail level = full
database:     ignore_bpf = no
database: using the "log" facility

        --== Initialization Complete ==--

  ______     -*> Barnyard2 <*-
 / ,,_  \    Version 2.1.14 (Build 337)
 |o"  )~|    By Ian Firns (SecurixLive): http://www.securixlive.com/
 + ''''' +   (C) Copyright 2008-2013 Ian Firns <firnsy@securixlive.com>

Using waldo file '/var/log/snort/barnyard2.waldo':
    spool directory = /var/log/snort
    spool filebase  = snort.u2
    time_stamp      = 1525234912
    record_idx      = 16
Opened spool file '/var/log/snort/snort.u2.1525234912'
Waiting for new data
```

Appendix F: ICMP summary detailed screen



Appendix G: Unique source and destination screen



Appendix H: Installation of Snort

Untitled - Notepad

File Edit Format View Help

```
sudo add-apt-repository ppa:ondrej/php
sudo apt-get update
sudo apt-get install -y apache2 libapache2-mod-php5.6 php5.6-mysql php5.6-cli php5.6
                        php5.6-common php5.6-gd php5.6-cli php-pear php5.6-xml

//
mkdir ~/snort_src

//installation of Data Aquistion Library
cd ~/snort_src
wget https://snort.org/downloads/snort/daq-2.0.6.tar.gz
tar -xvzf daq-2.0.6.tar.gz
cd daq-2.0.6
./configure
make
sudo make install

//installation of snort
cd ~/snort_src
wget https://snort.org/downloads/snort/snort-2.9.9.0.tar.gz
tar -xvzf snort-2.9.9.0.tar.gz
cd snort-2.9.9.0
./configure --enable-sourcefire
make
sudo make install
```

Appendix I: Configuring snort to run as NIDS

```
# Create the Snort directories:
sudo mkdir /etc/snort
sudo mkdir /etc/snort/rules
sudo mkdir /etc/snort/rules/iplists
sudo mkdir /etc/snort/preproc_rules
sudo mkdir /usr/local/lib/snort_dynamicrules
sudo mkdir /etc/snort/so_rules

# Create some files that stores rules and ip lists
sudo touch /etc/snort/rules/iplists/black_list.rules
sudo touch /etc/snort/rules/iplists/white_list.rules
sudo touch /etc/snort/rules/local.rules
sudo touch /etc/snort/sid-msg.map

# Create our logging directories:
sudo mkdir /var/log/snort
sudo mkdir /var/log/snort/archived_logs

# Adjust permissions:
sudo chmod -R 5775 /etc/snort
sudo chmod -R 5775 /var/log/snort
sudo chmod -R 5775 /var/log/snort/archived_logs
sudo chmod -R 5775 /etc/snort/so_rules
sudo chmod -R 5775 /usr/local/lib/snort_dynamicrules

# Change Ownership on folders:
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
```

Appendix J: Installing Barnyard2 and connecting it to MySQL DB

Untitled - Notepad

File Edit Format View Help

```
//installation of Barnyard2 pre-requisites
sudo apt-get install -y mysql-server libmysqlclient-dev mysql-client autoconf libtool

//installing Barnyard2

cd ~/snort_src
wget https://github.com/firnsy/barnyard2/archive/master.tar.gz -O barnyard2-Master.tar.gz
tar zxvf barnyard2-Master.tar.gz
cd barnyard2-master
autoreconf -fvi -I ./m4

//telling Banyard where to locate mysql libraries are

./configure --with-mysql --with-mysql-libraries=/usr/lib/x86_64-linux-gnu

//finishing installation

make
sudo make install


//Configuring MySQL and Setting up DB

 mysql -u root -p
mysql> create database snort;
mysql> use snort;
mysql> source ~/snort_src/barnyard2-master/schemas/create_mysql
mysql> CREATE USER 'snort'@'localhost' IDENTIFIED BY 'snort';
mysql> grant create, insert, select, delete, update on snort.* to 'snort'@'localhost';
mysql> exit
```

Appendix K: Installing PulledPork

Untitled - Notepad

File Edit Format View Help

```
//installation of Barnyard2 pre-requisites

sudo apt-get install -y libcrypt-ssleay-perl liblwp-useragent-determined-perl

//installing PulledPork

cd ~/snort_src
wget https://github.com/shirkdog/pulledpork/archive/master.tar.gz -O pulledpork-master.tar.gz
tar xzvf pulledpork-master.tar.gz
cd pulledpork-master/

sudo cp pulledpork.pl /usr/local/bin
sudo chmod +x /usr/local/bin/pulledpork.pl
sudo cp etc/*.conf /etc/snort
```

Appendix L: BASE installation and Setup

Untitled - Notepad

File Edit Format View Help

```
//installation and setup of BASE

This comes after installation of PERL and Apache2


//installing of ADODB

cd ~/snort_src
wget https://sourceforge.net/projects/adodb/files/adodb-php5-only/adodb-520-for-php5/adodb-5.20.8.tar.gz
tar -xvzf adodb-5.20.8.tar.gz
sudo mv adodb5 /var/adodb
sudo chmod -R 755 /var/adodb

//Downloading BASE to Apache folder and configuring base

cd ~/snort_src
wget http://sourceforge.net/projects/secureideas/files/BASE/base-1.4.5/base-1.4.5.tar.gz
tar xzvf base-1.4.5.tar.gz
sudo mv base-1.4.5 /var/www/html/base/

cd /var/www/html/base
sudo cp base_conf.php.dist base_conf.php

sudo vi /var/www/html/base/base_conf.php

//Filed manipulated, the # indicates the lines number

$BASE_urlpath = '/base';                    # line 50
$DBlib_path = '/var/adodb/';                #line 80
$alert_dbname     = 'snort';                # line 102
$alert_host       = 'localhost';
$alert_port       = '';
$alert_user       = 'snort';
$alert_password   = 'snort';   # line 106
```