ANOMALY DETECTION FOR NETWORK TRAFFIC DATA

BY

AGU JUSTICE CHIJINDU

REG. NO.: PG/MENG/15/76846

DEPARTMENT OF ELECTRONIC ENGINEERING,

FACULTY OF ENGINEERING

UNIVERSITY OF NIGERIA, NSUKKA

OCTOBER, 2018

TITLE PAGE

ANOMALY DETECTION FOR NETWORK TRAFFIC DATA

BY

AGU JUSTICE CHIJINDU

REG. NO.: PG/MENG/15/76846

DEPARTMENT OF ELECTRONIC ENGINEERING,

FACULTY OF ENGINEERING

UNIVERSITY OF NIGERIA, NSUKKA

OCTOBER, 2018

APPROVAL PAGE

ANOMALY DETECTION FOR NETWORK TRAFFIC DATA

BY

AGU JUSTICE CHIJINDU
REG. NO.: PG/MENG/15/76846

A RESEARCH PROJECT
FULFILLMENT OF THE REQUIREMENTS FOR THE AWARD OF MASTER OF ELECTRONIC ENGINEERING (DIGITAL ELECTRONICS AND COMPUTERS OPTION),
IN THE DEPARTMENT OF ELECTRONIC ENGINEERING,
UNIVERSITY OF NIGERIA, NSUKKA

AGU JUSTICE CHIJINDU (STUDENT)	SIGNATURE_____	DATE_____
ENGR. DR. O. N. ILOANUSI (SUPERVISOR)	SIGNATURE_____	DATE_____
PROF. E.C. OKAFOR (EXTERNAL EXAMINER)	SIGNATURE_____	DATE_____
ENGR. DR. C. C. UDEZE (HEAD OF DEPARTMENT)	SIGNATURE_____	DATE_____
ENGR. DR. C. A. MGBEMENE (CHAIRMAN, FACULTY POSTGRADUATE COMMITTEE)	SIGNATURE_____	DATE_____

CERTIFICATION

This is to certify that AGU JUSTICE CHIJINDU, a postgraduate student of the Department of Electronic Engineering, University of Nigeria, Nsukka with registration number PG/MENG/15/76846 has satisfactorily completed the requirements for the research work for the award of Master degree in Electronic Engineering (Digital Electronic and Computer Specialization).

_____	_____
ENGR. DR. O. N. ILOANUSI	ENGR. DR. C. C. UDEZE
(PROJECT SUPERVISOR)	(HEAD OF DEPARTMENT)

_____

ENGR. DR. C. A. MGBEMENE

(CHAIRMAN, FACULTY POSTGRADUATE COMMITTEE)

DECLARATION

I, AGU JUSTICE CHIJINDU, a postgraduate student of the department of Electronic Engineering, University of Nigeria, Nsukka, declare that the work embodied in this dissertation is original and has not been submitted by me in part or in full for any other diploma or degree of this University or any other Universities.

_____ _____

AGU JUSTICE CHIJINDU DATE
PG/MENG/15/76846

DEDICATION

This project is dedicated to almighty God, whose plan for my life is unchangeable.

ACKNOWLEDGEMENT

To the extent that I may have been successful in my stated aims and objectives of carrying out this project, the achievement so far would not have been easy without the support of many friends of mine, my family and the University staffs. Particularly, I would like to immensely thank my supervisors Dr. O. N. Iloanusi for her guidance and directions.

I would want to also thank my parents Rev and Mrs Edwin Agu and my brother Mr. Gabriel Agu for their love and continuous support throughout the program.

I want to thank my Uncle Mr Nicholas I. Udeh and His Wife Mrs Susan Udeh for their love and sponsorship throughout this program. Thanks to my beloved friends Ezema Celestine Anaenechi, Nkpozi Kelechi and all my class mates for their care and love.

ABSTRACT

The research work effectively produces a functional algorithm for detecting anomalous traffic data in network Transport Control Protocols (TCP). This research work proposed 'Synchronize' Synchronization Packet Flood Distributed Denial of Service (SYN Flood DDoS) attacks detection algorithm on network Transport Control Protocols (TCP), in order to analyze and examine network traffic traces and see how this affect detecting anomalies in distributed attacks. This anomaly detection algorithm was developed using Object-Oriented Software Engineering (OOSE) methodology, which is compliant to Model-Driven Architecture (MDA) in the development processes. The algorithm deploys Entity-Relationship model in organizing the main information object. Java programing language was chosen for the implementation of the proposed algorithm. Finally, Detection of SYN Flood Distributed Denial of Service attacks were performed on network packet traces (datasets) captured from 15th through 24th November, 2017 to determine how well anomalies are detected on TCP network protocols. The results obtained while testing the proposed algorithm summarizes SYS Flood DDoS attacks detected in each of the network packet traces. It can be observed that the attacks were detected in only four datasets, which are network packet traces, captured on 15th, 16th, 22nd and 24th November 2017, while other six datasets were attack free. All the attacks detected occurred in less than 1 minute. The result obtained in this experiment using the proposed JLP(Java Logical Program) SYN Floods DDoS attack detection algorithm, shows that SYN floods attack inundate a host server with [SYN] segment containing forged (spoofed) IP source addresses with non-existent or unreachable addresses. Host server responds with [SYN, ACK] segments to these addresses and then waits for responding acknowledgement [ACK] segments. Host server will not receive any acknowledgment [ACK] response and eventually time out. This is because the response was sent to non-existent or unreachable IP addresses. JLP algorithm continues to detect those attacks flooding a host with incomplete TCP connection ([SYN] and [SYN, ACK] without an [ACK]), the detection algorithm result shows that the attacker eventually attempts filling the memory buffer of the victim. Because once this buffer is full, the host can no longer process new TCP connection requests.

Table of Contents

LISTS OF FIGURES

LIST OF TABLES

LISTS OF ABBREVIATIONS

1.	AAA - - - -	Authentication, Authorization, and Accounting
2.	ACK - - - -	Acknowledgement
3.	CPU - - - -	Central Processing Unit
4.	COF - - - -	Connectivity Outlier Factor
5.	CSP - - - -	Cloud Service Provider
6.	CSS - - - -	Cross Site Scripting
7.	DB - - - -	Distance Based
8.	DCNs - - - -	Data Center Networks
9.	DDoS - - - -	Distributed Denial of Service
10.	DNS - - - -	Domain Name Server
11.	DOS - - - -	Denial of Service
12.	DPI - - - -	Deep Packet Inspection
13.	DIDMA - - - Mobile Agents	Distributed Intrusion Detection System Using
14.	ER - - - -	Entity-Relationship
15.	ERD - - - -	Entity Relation Diagram
16.	FIN - - - -	Finish
17.	GCCIDS - - - System	Grid and Cloud Computing Intrusion Detection
18	GUI - - - -	Graphical User Interface
19.	ICMP - - - -	Internet Control Message Protocol
20.	IDS - - - -	Intrusion Detection System
21.	IP - - - -	Internet Protocol
22.	IDE - - - -	Integrated Development Environment
23.	JLP - - - -	Java Logical Program
24.	LOF - - - -	Local Outlier Factor
25.	LBNL - - - -	Internal Enterprise Traffic

26.	MDA	-	Model-Dreiven Architecture
27.	MA	-	Mobile Agent
28.	MSS	-	Maximum Segment Size
29.	NNs	-	Neural Networks
30.	NIDS	-	Network Intrusion Detection Systems
31.	OOSE	-	Object-Oriented Software Engineering
32.	SYN	-	(synchronization) packet
33.	SYN/ACK	-	(synchronization acknowledged) packet
34.	SVM	-	Support Vector Machines
35.	TCP	-	Transport Control Protocols
36.	UDP	-	User Datagram Protocol
37.	UML	-	Unified Modelling Language
38.	VMs	-	Virtual Machines
39.	VMI	-	Virtual Mobile Infrastructures

CHAPTER ONE
INTRODUCTION

1.1 BACKGROUND OF STUDY

Ever since the computer and the critical data it holds came into headlines, so did the malicious programs, attacks and the threat landscape. There are thousands of cases of malware infection, zombies and trojans taking over networks in fast pace. The amount of data that passes through any switch, router, Firewall is enormous. There are 'gigabits of traffic' flowing every second through perimeter and internal networking devices. To protect this vast amount of data, designers have deployed host as well as network level controls and software. Every security measure deployed makes it one step harder for the attackers to gain access to the internal resources. There are devices that check for malware patterns, do heuristic scans, and find patterns that resemble a blacklisted file or a cyber-threat. This technology is termed as Deep Packet Inspection (DPI) for it inspects the payload as well as the protocol details of every packet against the set of signatures to match. But, with the constant evolution of attack vectors it's now a crazy fire-fighting exercise to match every type and strain of malware or every style and patterns of an attack. Moreover the detection capability bridge between malicious and benign software is shrinking rapidly. It is very important to have something that can help narrow down the spread of a malicious file. Traffic Anomaly is anything which is not expected in day-to-day traffic; something that creates an anomaly and raises an alarm. It can be huge amount of requests, response, particular TCP flag, DNS queries, anything.

1.2 PROBLEM STATEMENT

Life cannot be imagined without security, as it is one of the basic needs. Similarly, computer security is also the heart of today's technological world. Many organizations have been looking to move their services and business processes to the cloud, and so there is need for employees to have access from remote locations as well as the increasing number of online transactions that promotes an internet solution [1]. Data protection is the most important security issue in Cloud environment. In the service provider's data center, protecting data privacy and managing compliance are critical by using encrypting and managing encryption keys of data in transfer to the cloud. Denial of Service (DOS) attacks is popular due to their ability to significantly affect networks. DOS, as the name implies, exhausts the resources of the target network by sending invalid traffic. A DOS attack when carried out by multiple

devices is known as a Distributed Denial of Service (DDOS) attack. DDOS is considered to be the biggest threat for networks. These problems are intended to be solved by effectively producing a functional algorithm for detecting anomalous traffic data in Transport Control Protocols (TCP) network.

1.3 AIM & OBJECTIVES OF THE WORK

The aim of this project is to produce a functional algorithm for detecting anomalous traffic data in network Transport Control Protocols (TCP). The objectives are

- to deploy Entity-Relationship model in organizing the main information object.
- to develop an application that is aimed at storing captured network traffic traces into a database, and performing detection of SYN Floods DDoS attacks automatically and display result in enhanced form using the proposed algorithm.
- to study the effect of SYN Floods Distributed Denial of Service (DDoS) anomaly in network Transport Control Protocols (TCP) traffic data.

1.4 SIGNIFICANCE OF THE STUDY

This research work is expected to reveal the possible variation in normal behaviour of Transport Control Protocol (TCP) network traffic data due to the presence of SYN Flood Distributed Denial of Service (DDoS) attacks. It will also develop an anomaly detection algorithm which is an important tool for detecting SYN Flood DDoS attack automatically. The validation of the proposed algorithm will compare the existing algorithms with the proposed algorithm in accordance with normal characteristics of TCP (Transport Control Protocols) network traffic packets connection establishment processes.

1.5 SCOPE OF THE STUDY

This research provided an extensive literature review on related technologies and existing works. It also produces a functional algorithm for detecting SYN Flood Distributed Denial of Service attack on network Transport Control Protocol traffic packet traces collected from WIDE MAWI WORKING GROUP repository directory which is publicly made available for researchers. WIDE MAWI WORKING GROUP has carried out network traffic measurement, analysis, evaluation, and verification from the beginning of the WIDE Project.

The proposed algorithm were developed such that it can store network traffic packet traces, detect SYN Flood Distributed Denial of Service attacks automatically in each of the network traffic packet traces(datasets) captured from 15th to 24th November 2017 and display the result in an enhanced form. Finally, results were obtained, and discussed; and conclusion was drawn from the result findings.

1.6 ORGANIZATION OF THE REPORT

Chapter One provides the background of study, Aim and objectives of the work, problem statement, significance of the study and scope of the study.

Chapter Two presents background information on existing research in the fields of network data centers, cloud computing architecture, cloud computing stack, data center traffic, traffic analysis, anomaly detection and anomaly detection techniques. This information is necessary to understand decisions taken throughout the course of this work.

Chapter Three provides directions which this project will take in order to achieve its objectives. The detailed information on the process involved in data acquisition, important attributes of the data. Also the information that the attributes gives about the data transfer and the nature of data transfer are discussed extensively under data analysis section.

Chapter Four provides details on the exact experiment scenarios that were conducted such as the development, implementation and testing processes. The results of these experiments are shown using plots and analyzed based on their behaviour.

Chapter Five completes this work by summarizing all the research studies during this project. Limitations and the key findings based on the presented results including suggestions for future work and improvements are all given.

CHAPTER TWO
LITERATURE REVIEW

2.1 OVERVIEW OF ANOMALY DETECTION TECHNIQUES IN CLOUD COMPUTING

This section introduces major topics related to this project that will help the reader understand decisions taken throughout it. This work involves the studies of network traffic in network data centers. In the First, data center traffic were explored, that is its complexity and behavior. The next section discusses aspects of traffic analysis in the infrastructure and how different data processing techniques affects the traffic. Traffic analysis is done to detect anomalies, and the chapter continues by presenting types of anomalies and different anomaly detection techniques. The last section of this chapter reviews existing literature on anomaly detection.

2.1.1 Datacenter Traffic

Data center traffic is composed of internal and external traffic. Internal traffic accounts for server to server requests (where both servers are hosted on the same data center), replication of existing servers, and more. According to [2], data centers play an increasingly important role in processing the large amount of information generated in today's society. According to [3], data Centers have become the backbone of the present digital world as they provide the computing capabilities and the storage required for the billions of devices that are connected to the internet. Cloud services delivered by datacenter networks yield new opportunities to provide protection against disasters. Cloud services require a network substrate with high capacity, low latency, high availability, and low cost, which can be delivered by optical networks. In such networks, path protection against network failures is generally ensured by providing a backup path to the same destination (i.e., a datacenter), which is link-disjoint to the primary path. This protection fails to protect against disasters covering an area which disrupts both primary and backup paths. Also, protection against destination (datacenter) node failure is not ensured by a generic protection scheme. Moreover, content/service protection is a fundamental problem in a datacenter network, as the failure of a datacenter should not cause the disappearance of a specific content/service from the network. According to [4], Data Center Networks (DCNs) provide the core infrastructure backbone for various small and large scale enterprise applications, including Cloud Computing applications. TCP

is used by a plethora of Internet applications today and it also retains its dominance in DCNs, in terms of the most widely deployed transport protocol. The nature of DCN traffic, however, largely differs from that of the Internet. Relying on TCP's congestion control opens up the network to denial of service attacks and performance interference. Datacenters of various sizes are being built and employed for a diverse set of purposes today. Datacenters are experiencing an exponential increase in the amount of network traffic that they have to sustain due to cloud computing and several emerging web applications. To face this network load, large data centers are required with thousands of servers interconnected with high bandwidth switches [5]. Datacenter networks carry background traffic consisting of long-lived flows, which involve large data transfers [6]. According to [7], Data centers employ virtualization to organize thousands of servers. Applications such as MapReduce and Hadoop are deployed in virtual machines (VMs). However, with the development of client requirements, the traffic among VMs, which host different types of applications, becomes irregular and unpredictable. This traffic may cause congestion links and performance violations in applications. External traffic is defined as flows that come from or are sent to somewhere outside the current data center. In [8], a detailed examination of evolving traffic characteristics, operator requirements, and network technology trends suggests a move away from nonblocking interconnects in data center networks (DCNs). As a result, recent efforts have advocated oversubscribed networks with the capability to adapt to traffic requirements on-demand. According to [9], Transport Control Protocol (TCP) incast congestion happens in high-bandwidth and low-latency networks when multiple synchronized servers send data to the same receiver in parallel. For many important data-center applications such as MapReduce and Search, this many-to-one traffic pattern is common. Hence TCP incast congestion may severely degrade their performances, e.g., by increasing response time. In [9], a network element in a data center includes a plurality of servers and a switch. The switch includes a plurality of physical ports, a packet-forwarding table, and an application program interface (API) for modifying a packet-forwarding behavior of the switch. According to [9], the packet-forwarding table determines a packet-traffic distribution across the servers by mapping packet traffic arriving at the switch to the plurality of ports. Each port of the plurality of physical ports is in communication with one of the servers. Some implement load balancing, as presented in [10], which cause traffic between any pair of servers to bounce off a randomly chosen switch in the top level. Others do not rely on randomization for load

balancing but use hash-based implementations [11]. Traffic is split per flow across multiple links. For a uniform distribution, a centralized controller can be used to reassign flows or distribute traffic based on packets. According to [12], the simplest solution is randomized load balancing where each flow is assigned a random path from a set of possible paths. However, random selection causes hot spots to develop; hence, they propose a different solution to split flows based on a hash of the five tuple in each packet. Another aspect of data center traffic is its behavior. Routing determines the path of flows only once the destination is known. Another paper discussing internal traffic behavior, [10], obtains conclusions based on data from five cloud data centers and compares it to traffic data from two private enterprise data centers and three university campus data centers. It states that in cloud data centers a majority of traffic originating from the servers (80%) stays within the rack, whereas for university and private datacenters most of the traffic leaves the rack. Another difference between a cloud data center and a private or university one is the amount of servers and devices in the data center. The cloud datacenters, which are part of the research in [10], have 5 times more servers and devices than a private data center, and 10 to 15 times the servers and devices of a university data center. These cloud data centers provide support for wide range of services such as MapReduce, messaging, webmail, and web portal. Each of these applications consists of dependencies deployed across the data center, and this application mix impacts the traffic results.

2.1.2 Traffic Analysis

The purpose of traffic analysis is to detect network traffic anomalies that could disrupt the normal operation of the network infrastructure. According to [13], anomaly detection or analysis typically operates on pre-processed traffic. This section shows different approaches on how and where traffic is analyzed. According to [14], traffic analysis is conducted at the transport level. Scanning traffic is removed before proceeding with analysis because it causes the magnitude of traffic to increase. [15] Says that analysis is done at the entry point of the data center.

In [16] they have coarse grained data for their experiments, such as average traffic volumes and loss rates. One of the data sets they use is collected from 19 data centers. To benefit from the coarse grained data, they aggregate traffic statistics in 5 minute increments. Also they approximate traffic at the edge switches by generating fine grained data from the coarse-

grained one. Their goal is to evaluate the effect of traffic mix and packet sampling on anomaly visibility. Traffic mix means that data is collected from different border routers. Packet sampling is reducing the amount of traffic data measured. Sampling at rate n means that every n^{th} packet from the original traffic data is inspected uniformly at random and its features are recorded. The purpose is to summarize the traffic data. However, sampled traffic is incomplete, because small flows are likely to be missed. This could affect count metrics, but the impact of packet sampling on entropy and volume metrics depends largely on the traffic mix and type of anomaly. The feature counts are well-suited for detection of anomalies that cause widening of the port or IP distribution, entropy metrics is more appropriate for detecting anomalies that cause a concentration of the distribution on a specific port or IP address. Packet sampling can also be done randomly by selecting each packet with a certain probability [17]. Each packet is selected with a probability q or discarded with probability 1 − q. It reduces the amount of data, but its effects have to be assessed before using sampled data. Probability based packet sampling can disturb flow counts as well as the uniformly distributed sampling because small flows are sampled with a lower probability than larger flows. Packets are pre-processed by sampling and temporal aggregation [17]. The latter means that all packets in a measurement interval are represented by their temporal mean. The goal is to transform the traffic trace to a desired granularity before it is observed for anomalies. A different idea on how to decrease amount of traffic analyzed is proposed by [18]. They suggest not analyzing all traffic in the data center but instead have switches and routers detect if there is a change in network behavior at their end, and use traffic from devices that have detected abnormal behavior. Then all this traffic will be aggregated and analyzed together. Another method for traffic analysis is to approach it as proposed in [19], and record packet sequences that match specified application specific signatures.

2.1.3 Anomaly detection

The first step of analyzing traffic is to establish a baseline of normal traffic. One way to create a baseline is to use the average over a past time period. Any deviations from this baseline behavior are called anomalies. Anomaly size is defined as the distance between sample view and corresponding baseline. Anomaly detection is the problem of finding patterns in data that do not follow expected behavior, i.e. differ from the baseline. However, the difference between normal and anomalous is often not precise [20]. Hence, there exist numerous methods that analyze that difference and classify it as an anomaly or not.

2.1.3.1 Taxonomy of Anomalies

Anomaly detection aspires at finding the presence of anomalous patterns in network traffic and usual detection of such outline can provide network administrator with extra information source to identify network behavior or tracing and locating the root cause of faults in a network [21]. Anomalies can be classified into three categories: Point Anomalies, Contextual Anomalies, and Collective Anomalies [22].

- **Point anomaly** - This is when an individual data instance deviates from its normal activity or form. It is said to be anomalous, because other data are normal. This shows that the anomalous activity lies outside the boundaries of the normal region. This is the easiest type of anomaly amongst the 3 types or categories and it is the strength or importance of anomaly detection. Figure 2.1 illustrates the point anomalies. From Figure 2.1, N_1 and N_2 are regions of normal behavior, Points O_1 and O_2 are anomalies and Points in region O_3 are anomalies. [21].

- **Contextual anomaly** - The contextual anomalies occur when the occurrence of information is or shows traces of anomalous character in an exact or precise context, which is the unwanted behavior of activities that surrounds an individual data instance. Figure 2.2 illustrates the Contextual Anomaly. As it is shown in Figure 2.2, when this occurs it is characterized as a related anomaly. This requires an idea or notion of context in the data instance. It is also referred to as conditional anomalies [21].

- **Collective anomaly** - This is when a collection of related data instances is anomalous with respect to the entire data set, it is termed as a collective anomaly. The individual data instances in a collective anomaly may not be anomalies by themselves, but their occurrence together as a collection is anomalous. Figure 2.3 illustrates the Collective anomaly [21].

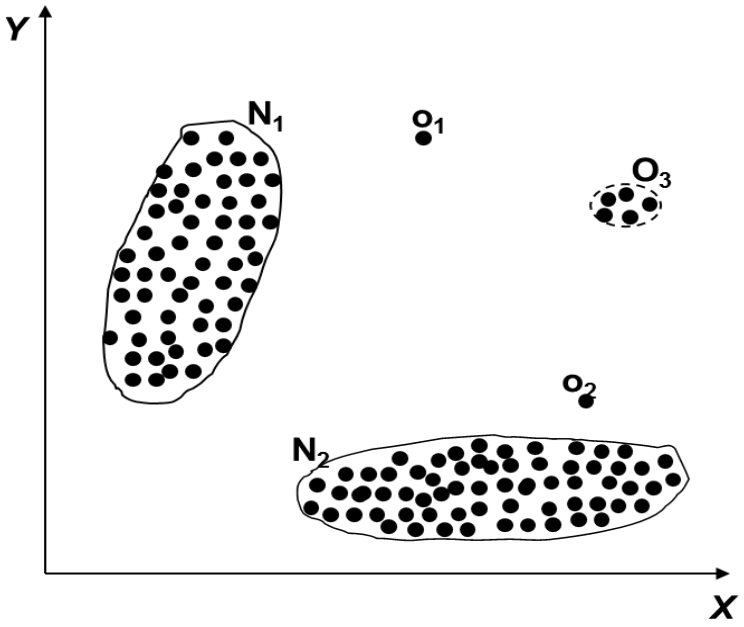


Figure 2.1: Illustration of point anomaly

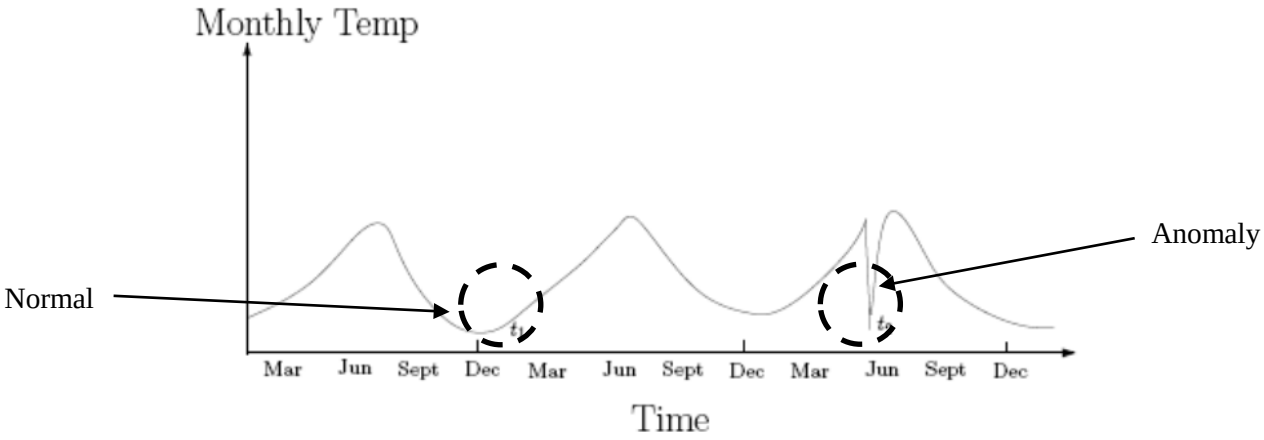


Figure 2.2: Illustration of contextual anomaly

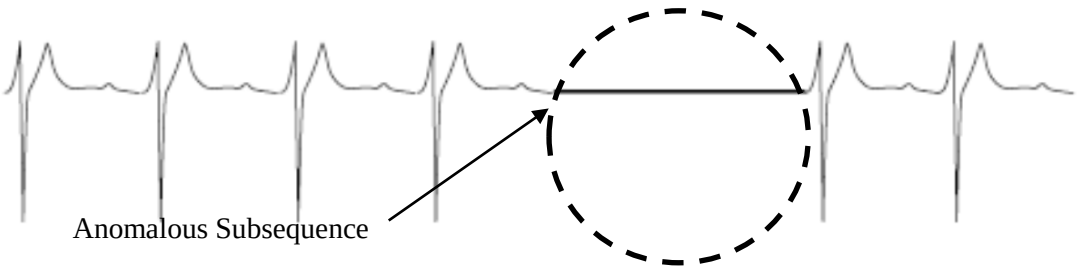


Figure 2.3: Collective anomaly

2.1.3.2 Distributed Denial of Service

A Distributed Denial of Service (DDoS) attack is an action that aims to make a network resource or service unavailable to its users. This is usually achieved by flooding the destination with multiple service requests such that the server overloads and crashes or all resources are consumed and the service becomes unavailable. In a Distributed Denial of Service (DDoS) attack, the attacker makes a huge impact on the victim by having multiplied power of attack derived by a large number of computer agents. It is made possible by the attacker through making a large number of computer machines under his control over the internet before applying an attack. In fact, these computers are vulnerable in the public network and the attacker exploits their weaknesses by inserting malicious code or some other hacking technique so that they become under the control of the attacker. These compromised machine scan be hundreds or thousands in numbers. They behave as agents of the attacker and are commonly termed as ‘zombies’. The entire group of zombies is usually named as a ‘botnet’. The size of the botnet decides the magnitude of attack. For larger botnet (increased number of zombies in a botnet), attack is more severe and disastrous. Within a botnet, the attacker chooses ‘handlers’ which perform command and control functions and pass the instructions of the attacker to the zombies. The zombies directly attack the victim. There is a group of zombies or agents under each handler. These handlers also pass the information received from zombies about the victim to the attacker [23]. Therefore, handlers are the machines which directly communicate with the attacker and zombies. As the handlers and zombies are also compromised machines in the public network under the control of an attacker, the users of such machines are usually unaware of the fact that there machines are being used as a part of some botnet. A typical architecture of DDoS attack is mentioned in *figure 2.4*.

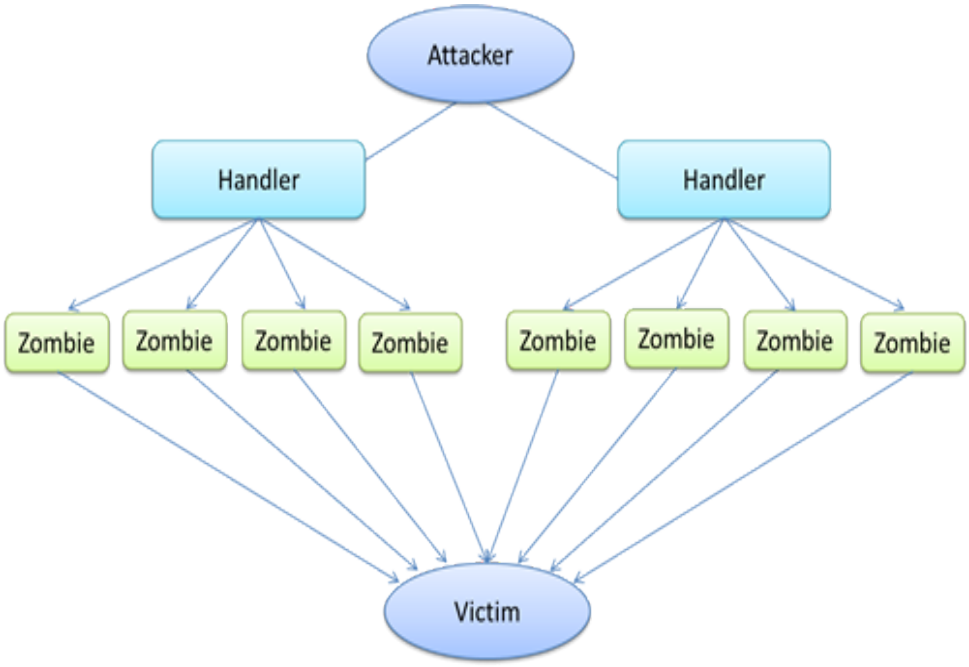


Figure 2.4: Architecture of DDoS Attack

The attack employs client server technology and a stream of data packets is sent to the victim for exhausting its services, connections, bandwidth etc. The data flood attack type of DoS is mostly used in DDoS attacks.

Classification of DDoS Attacks

With the evolution of internet, cyber-attacks have also increased manifold. Earlier DDoS attacks were manual, where attacker had to perform many steps before the launch of final attack, such as port scanning, identifying available machines in the public network to create botnet, inserting malware etc.

With the passage of time, sophisticated attack tools have been developed to assist attackers in performing all or some steps automatically to reduce human effort. The attackers can just configure desired attack parameters and the rest is done by the automated tools. Some common automated attack tools available are Trinoo, TFN (Tribe Flood Network), TFN2K, Stacheldraht, Shaft, Knight and Trinity. Some of them work on IRC (Internet Relay Chat) where handlers and zombies do not know identities of each other and the communication among them is done indirectly. The others are agent based in which communication is direct and handlers and zombies know each other's identity [24]. Therefore, when DDoS attacks are

classified by the degree of automation, they are mentioned as Manual, Semi-automatic and Automatic attacks.

DDoS attacks are further classified by attack rate dynamics i.e. how rate of attack varies with respect to the passage of time. The classes are Continuous Rate and Variable Rate attacks. In continuous rate, the attack has constant flow after it is executed. On the other hand, variable rate attack changes its impact and flow with time, making it more difficult to detect and respond to. Within variable rate, the attack rate dynamics can further be implemented as Fluctuating or Increasing. Moreover, based on the data rate of attack traffic in a given network, the attacks are also categorized as high rate and low rate DDoS attacks [25]. DDoS attacks are also classified in the literature as 'by impact' i.e. it can be Disruptive in which the normal service is completely unavailable to users, or it can be Degrading in which the service is not completely unavailable but experiences considerable decrease in the productivity.

The major classification of DDoS attacks is 'by exploited vulnerability' through which the attacker launches an attack on the victim. The classification is given in Figure 2.5. In the said classification, flood attack is used to bring down the victim's machine or network's bandwidth. It has a few major sub-classes like User Diagram Protocol (UDP) flood, Internet Control Message Protocol (ICMP) flood and TCP flood. In fact, all flooding attacks generated through DDoS can be of two types; direct attacks and reflector attacks [26]. In direct attacks, zombie machines directly attack the victim as shown in the attack architecture in Figure 2.6 shows reflector attacks. On the other hand, in reflector attacks, zombies send request packets with spoofed IP (IP of the victim) in source address to a number of other compromised machines (PCs, routers etc.) and the reply generated from such machines is targeted towards the victim for the impact desired by the attacker. In such a way, reflection of the traffic is observed in these kinds of attacks. A classic example is sending 'ping' requests with spoofed source IP and the 'ping' replies are targeted towards the victim. The goal of attacker launching such attacks is to saturate the bandwidth of the victim with huge amount of traffic. The architecture of reflector attack is shown in figure 2.6.

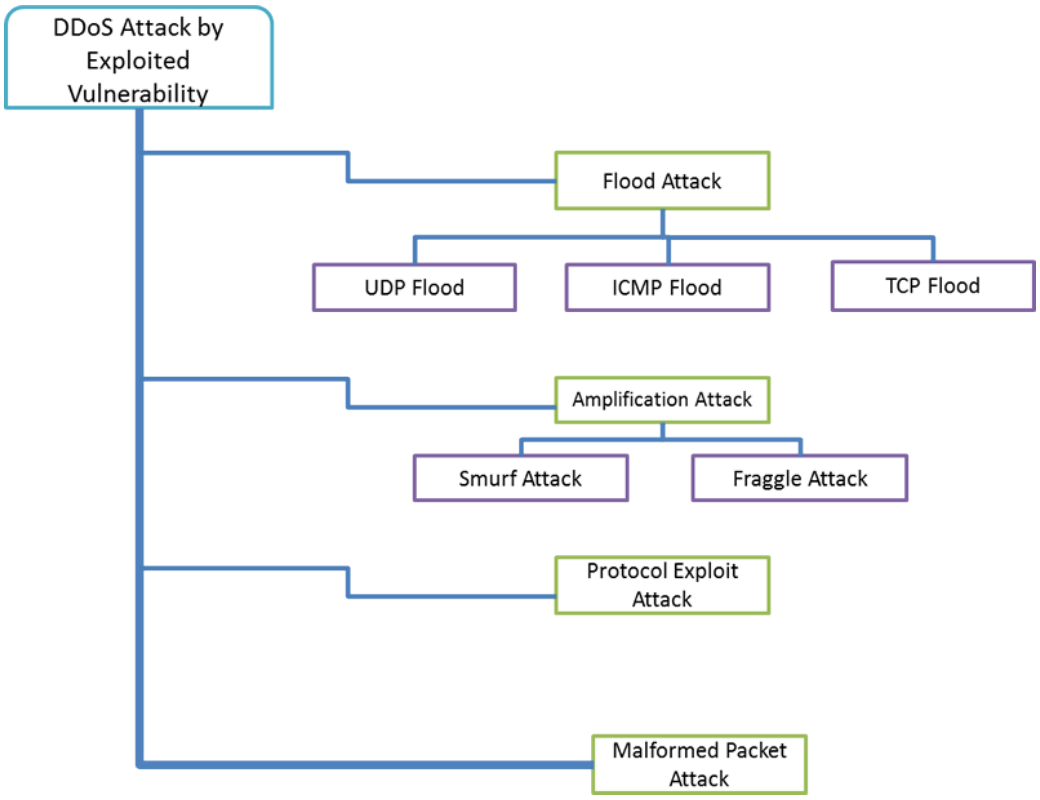


Figure 2.5: DDoS attack by exploited vulnerability - Classification

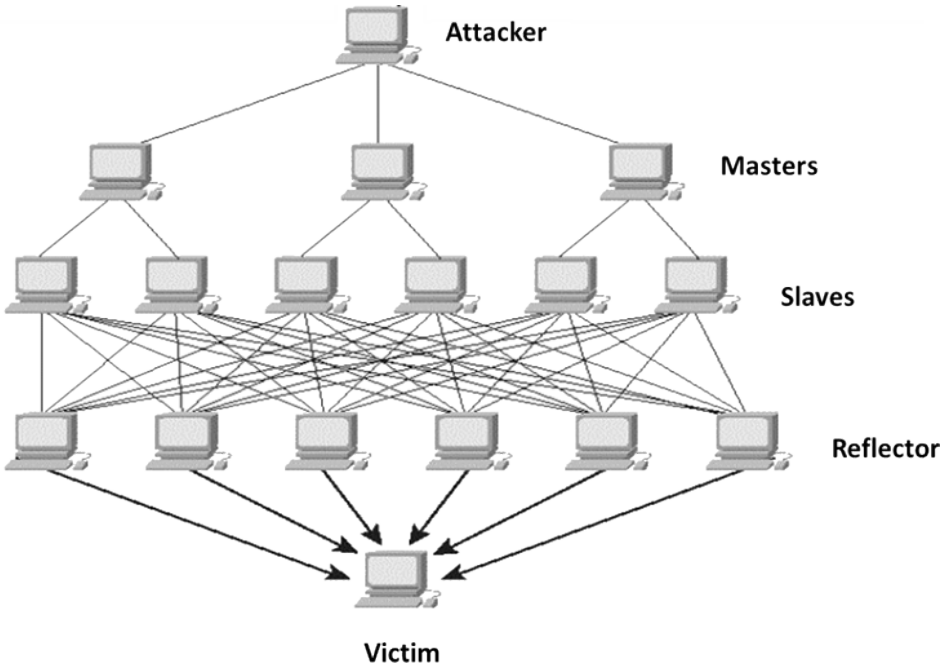


Figure 2.6: Architecture of DDoS reflector attack (‘Masters’ represent Handlers and ‘Slaves’ represent Zombies)

In UDP (User Datagram Protocol) flood attack, the target is usually the victim's machine. The agents of botnet send huge amount of UDP packets to the victim with randomly selected destination port. The source IP is spoofed by the attacker to prevent its own machine(s) from any return effect or trace back, therefore the reply is not reached to actual traffic generated sources. When the victim's machine is continuously made busy to identify ports and send reply messages at a very fast rate i.e. beyond its processing speed and capacity, it crashes or is brought down. Moreover, the huge amount of UDP packets sent by the attack sources can also lead congestion in victim's bandwidth and degrade services for other legitimate requests that may be sent to other machines on the same network.

In Internet Control Message Protocol (ICMP) flood attack, the target is bandwidth saturation. In this attack, huge amount of echo packets i.e. 'ping' requests are sent by the attack sources to remote host(s). The source IP is spoofed and contains victim's address on targeted network. As a result, massive traffic is generated in the network which ultimately leads to the bandwidth saturation. The 'ping' requests can be sent directly or through agents to multiply the effect.

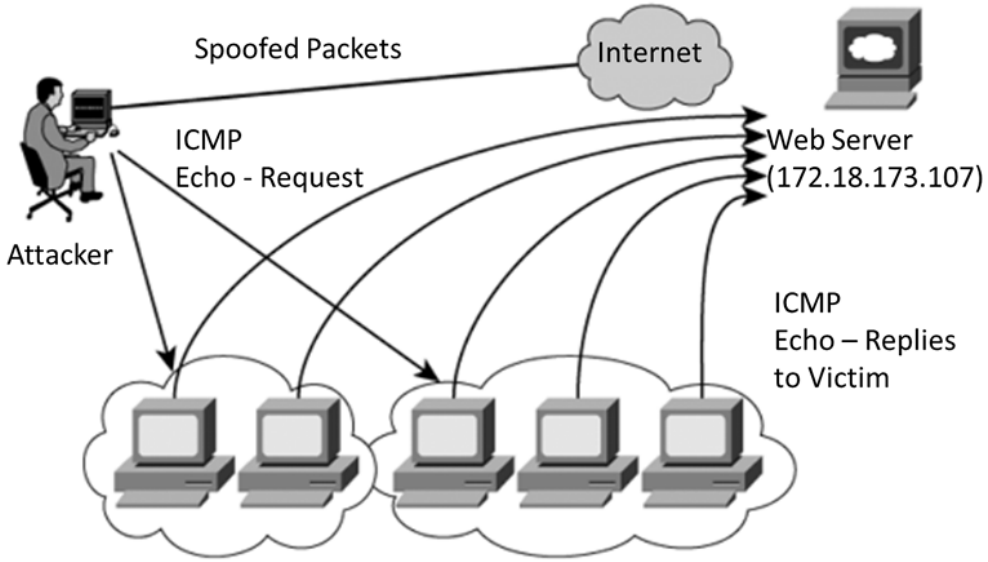


Figure 2.7: ICMP flood attack

In Figure 2.7, it is shown that an attacker spoofs IP packets before sending ICMP-ECHO-REQUEST or 'ping' packets to remote hosts. The hosts then generate ICMP-ECHO-REPLY packets to respond to the spoofed source on targeted network resulting in the bandwidth saturation.

In TCP flood attack, sophisticated attackers generate TCP traffic with legitimate-like packet headers so that traffic is not easily detectable as an attack. The payload is formed with random values and huge amount of such traffic is sent towards the victim for exhausting its services, bandwidth and connection [26]. In amplification attacks, the broadcast feature of IP addresses is exploited on network routers. The attack is generated with spoofed source IP addresses so that routers broadcast the same within their broadcast domains to update routing tables. In this way, amplification and reflection of IP traffic are observed as all routers broadcast spoofed IP addresses to all addresses in their broadcast domain. As a result, massive traffic is generated in the network reducing the bandwidth for legitimate requests. Two major classes of amplification attacks are smurf attack and fraggle attack. They are effectively the same as ICMP flood attack and UDP flood attack respectively and work through sending ICMP echo and UDP echo packets to bring down a victim or saturate bandwidth with the help of spoofed source IP addresses.

The protocol exploit attacks make use of some weakness of a protocol. A common example is TCP SYN attacks which exploit three way handshake feature of Transmission Control Protocol. In this client / server model, the client first initiates communication by sending a SYN signal to the server and requesting to establish a connection. The server responds with ACK signal which is an acknowledgement that the server is ready to establish the requested connection. Finally, the server waits for ACK signal from the client and when it receives the same, connection is successfully established.

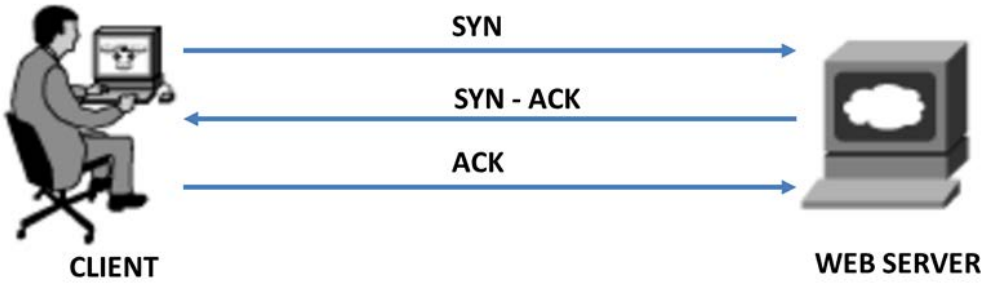


Figure 2.8: Three-way handshakes in TCP

The SYN ACK attack or SYN flood attack is generated by sending a large number of spoofed SYN signals to the victim and never acknowledging the same for which the server waits after sending ACK signal to the client. The server has to wait for a certain period of time before it

releases the connection for any new request (normally it is between 45 to 360 seconds) [25]. The buffer capacity is limited for connections and if large number of such attack based SYN messages is sent through multiple agents to occupy the space in buffer, it results in a full queue buffer which makes the server unable to process new legitimate requests. Moreover, if the server is to maintain full queue buffer all the time and high quantity of resources is consumed in the process, it may give a rise to TCP / IP stack overflow leading the server to crash. It is also considered under the category of flood attacks.

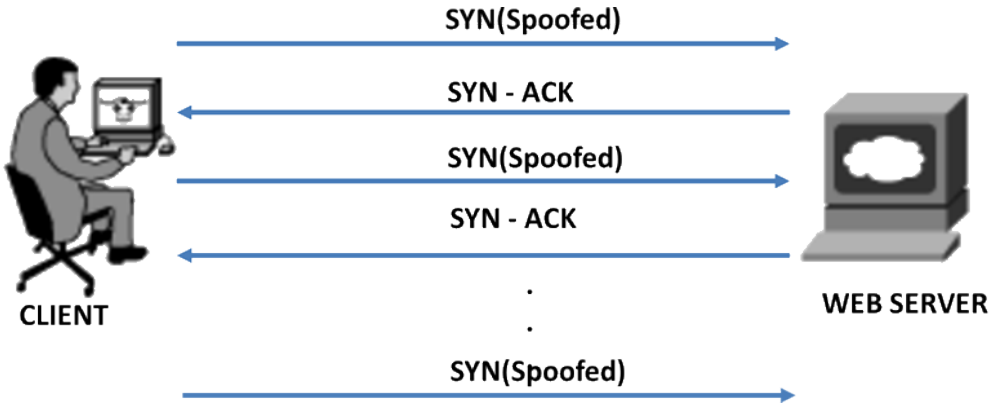


Figure 2.9: SYN ACK attack in TCP

In malformed packet attacks, the attacker relies on malicious data within IP packets that are sent by agents to the victim. These attacks can completely crash the victim machine. Two subcategories of such attacks are IP address attack and IP packet options attack. In IP address attack, the packets are formed with same source and destination address. As a result, the victim machine is unable to process such packets and tricked in a way that it can finally crash. On the other hand, in IP packet options attack, the optional fields in IP packets are randomized by the attacker to trick the processing of victim machine. For example, all quality of service bits is made '1' for which the victim is unable to extract the information from packets and the system speed is greatly reduced. When this attack is applied with different combinations through multiple agents, it may also lead a victim machine to crash.

Application Layer DDoS Attacks

In the network layer or infrastructure layer (Layer3) attacks, the malicious part resides in packet header or payload to compromise victim's CPU cycles, processing, bandwidth etc. However, with the introduction of sophisticated DDoS detection & mitigation tools, attackers have also started changing their strategies to avoid detection and mitigation by increasing

their focus towards application layer (Layer 7) attacks. These attacks mimic the legitimate clients to disturb or destroy the victim's resources. Therefore, traditional DDoS detection techniques are unable to identify such attacks. In these attacks, complete communication with the victim is established just like legitimate users. However, numerous connections are generated aiming to deny or degrade the service or bandwidth for legitimate clients.

Application layer attacks are subject to the establishment of complete TCP connections with the victim. Therefore, the attacker has to disclose real IPs of zombie machines to the victim. Otherwise, it is not possible to make such connections.

However, due to large number of zombies, the attacker does not worry about this attack limitation [26]. If such machines are identified and filtered at some stage, the attacker uses other group or pool of zombies to process the continuity of the attack. After establishing TCP connections with the victim in a large number, the attacker starts communication through sending requests for relatively large processing such as downloading heavy image files or making database queries. In this way, resources are reserved against such attack traffic to deny or degrade the services for legitimate users. Effectively, application layer attacks are also flooding attacks and categorized as HTTP flood, HTTPS flood, FTP flood etc. Sometimes, they are collectively mentioned as GET floods.

2.1.3.3 Port Scans

A port scan is an attack that sends requests to a range of server ports looking for an active port, which can be used to exploit a known vulnerability of the service. In [27], they classified the scanning attacks into three categories: "vertical", "horizontal" and "mixture of horizontal and vertical". First type refers to a scanner looking for open ports on a single target destination IP (scanner scans various ports for a single IP and finds the possibility of some open ports for that IP); while the second one refers to a scanner looking for one specific open port on several target machines. For example, scanner scans http port 80 for every IP present in the network. The "mixture" scan type refers to a scanner checking fixed range of ports on a specific set of destination machines.

2.1.4 Anomaly Detection Techniques

In Figure 2.10 illustrates the taxonomy of Anomaly detection techniques. This taxonomy of anomaly detection techniques are detailed in this section.

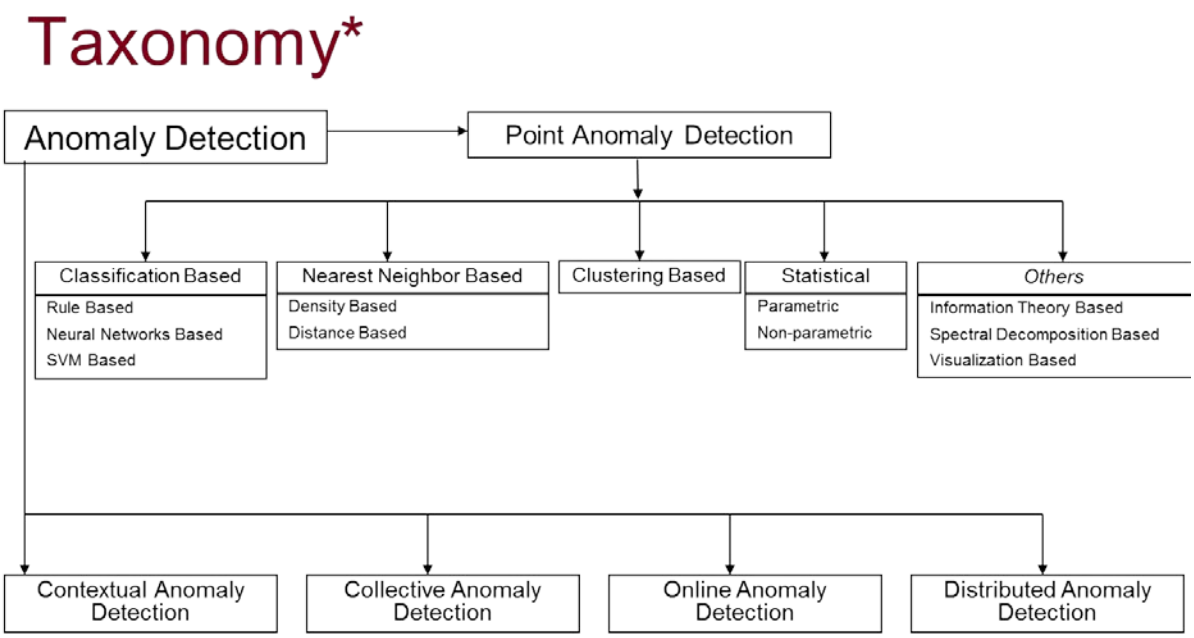


Figure 2.10: Taxonomy of AD Techniques

2.1.4.1 Classification Based Techniques

Main idea of Classification Based Techniques as show in Figure 2.10, is to build a classification model for normal (and anomalous (rare)) events based on labelled training data, and use it to classify each new unseen event. Secondly; classification models must be able to handle skewed (imbalanced) class distributions.

The categories of Classification Based Techniques are as follows:

- Supervised classification techniques: Require knowledge of both normal and anomaly class and also build classifier to distinguish between normal and known anomalies.
- Semi-supervised classification techniques: Require knowledge of normal class only and Use modified classification model to learn the normal behavior and then detect any deviations from normal behavior as anomalous.

The advantages of Classification Based Techniques categories are as follows:

Supervised classification techniques are models that can be easily understood and also have high accuracy in detecting many kinds of known anomalies.

Semi-supervised classification techniques are models that can be easily understood and Normal behavior can be accurately learned

The drawbacks of Classification Based Techniques categories are as follows:

Supervised classification techniques requires both labels from both normal and anomaly class and cannot detect unknown and emerging anomalies

Semi-supervised classification techniques: Require labels from normal class and possible high false alarm rate - previously unseen (yet legitimate) data records may be recognized as anomalies

A. Rule Based Techniques are as follows:

- Creating new rule based algorithms (PN-rule, CREDOS)
- Adapting existing rule based techniques includes Robust C4.5 algorithm and Adapting multi-class classification methods to single-class classification problem
- Association rules are rules with support higher than pre specified threshold may characterize normal behaviour. Anomalous data record occurs in fewer frequent item sets compared to normal data record and frequent episodes for describing temporal normal behavior
- Case specific feature/rule weighting which includes: Case specific feature weighting, and Decision Tree Learning. For the Decision tree learning, each rare class test example should be replaced with global weight vector. Global weight vector contains dynamically generated weight vector that depends on the path taken by the rare class test example. Also case specific rule weighting - LERS (Learning from Examples based on Rough Sets) algorithm increases the rule strength for all rules describing the rare class.

B. Neural Networks Based Techniques are follows: Multi-layer perceptron. This includes measuring the activation of output nodes, and extending the learning beyond decision boundaries. Decision boundaries encompass the following: Equivalent error bars as a measure of confidence for classification; Creating hyper-planes for separating between various classes, but also to have flexible boundaries where points far from them are outliers; Auto-associative neural networks, such as replicator NNs and Hopfield networks; Adaptive Resonance Theory based; Radial Basis Functions based, such as adding reverse connections from output to central layer, which then allows each neuron to have associated normal distribution, and any new instance that does not fit any of these distributions is an anomaly; and Oscillatory networks. Note that relaxation time of oscillatory NNs is used as a criterion for novelty detection when a new instance is presented.

C. Support Vector Machines (SVM) Techniques

- ❖ SVM Classifiers
- ❖ Main idea are such that the normal data records belong to high density data regions, Anomalies belong to low density data regions, it uses unsupervised approach to learn high density and low density data regions and also uses SVM to classify data density level. Also Data records are labelled (normal network behavior vs. intrusive) and use standard SVM for classification.

2.1.4.2 Nearest Neighbor Based Point Anomaly Detection Techniques

The key assumption here is such that normal points have close neighbors while anomalies are located far from other points. General two-step approach includes computing neighborhood for each data record and analyzing the neighborhood to determine whether data record is anomaly or not.

The following are the categories of Nearest Neighbor Based Point Anomaly Detection Techniques:

- ❖ Distance based methods means that Anomalies are data points most distant from other points.

- ❖ Density based methods means that Anomalies are data points in low density regions.

The advantages of Nearest Neighbor Based Point Anomaly Detection Techniques are as follows:

It can be used in unsupervised or semi-supervised setting (do not make any assumptions about data distribution).

The drawbacks of Nearest Neighbor Based Point Anomaly Detection Techniques are as follows:

If normal points do not have sufficient number of neighbors the techniques may fail**,** it is computationally expensive, in high dimensional spaces, data is sparse and the concept of similarity may not be meaningful anymore. Due to the sparseness, distances between any two data records may become quite similar, each data record may be considered as potential outlier!

A. Distance based approaches is as follow

A point O in a dataset is a DB (p, d) outlier if at least fraction p of the points in the data set lies greater than distance d from the point O

B. Nearest Neighbor (NN) approach are as follows

For each data point d compute the distance to the k-th nearest neighbor d_k. Sort all data points according to the distance d_k, Outliers are points that have the largest distance d_k and therefore are located in the more sparse neighborhoods, Usually data points that have top n% distance d_k are identified as outliers such as n – user parameter, and it is not suitable for datasets that have modes with varying density.

C. The density based approaches are as follows

It computes local densities of particular regions and declares instances in low density regions as potential anomalies. Approaches include Local Outlier Factor (LOF), Connectivity Outlier Factor (COF) and Multi-Granularity Deviation Factor (MDEF).

2.1.4.3 Clustering Based Point Anomaly Detection Techniques

The Key assumption for clustering based point anomaly detection techniques are such that the normal data records belong to large and dense clusters, while anomalies belong do not belong to any of the clusters or form very small clusters. It is categorized according to labels which includes Semi-supervised – cluster normal data to create modes of normal behavior. If a new instance does not belong to any of the clusters or it is not close to any cluster, is anomaly and Unsupervised – post-processing is needed after a clustering step to determine the size of the clusters and the distance from the clusters is required from the point to be anomaly.

Anomalies detected using clustering based methods can be data records that do not fit into any cluster (residuals from clustering), Small clusters and Low density clusters or local anomalies (far from other points within the same cluster).

The advantages of clustering based point anomaly detection techniques are as follows:

It has no need to be supervised and it can be easily adaptable to online / incremental mode suitable for anomaly detection from temporal data.

The drawbacks of clustering based point anomaly detection techniques are as follows:

It is computationally expensive in the sense that using indexing structures (k-d tree, R* tree) may alleviate this problem, If normal points do not create any clusters the techniques may fail and In high dimensional spaces, data is sparse and distances between any two data records may become quite similar, meaning that clustering algorithms may not give any meaningful clusters.

2.1.4.4 Statistical Based Point Anomaly Detection Techniques

Data points are modelled using stochastic distribution: the points are determined to be outliers depending on their relationship with this model.

The advantage of Statistical Based Point Anomaly Detection Techniques is as follows:

It utilizes existing statistical modelling techniques to model various types of distributions.

Challenges of Statistical Based Point Anomaly Detection Techniques are as follows:

With high dimensions, it is difficult to estimate distributions, and secondly parametric assumptions often do not hold for real data sets.

Types of Statistical Techniques

The following are the types of statistical based anomaly detection techniques:

A. Parametric Techniques

As illustrated in figure 2.10, parameter technique assumes that the normal (and possibly anomalous) data is generated from an underlying parametric distribution. Learn the parameters from the normal sample and also determine the likelihood of a test instance to be generated from this distribution to detect anomalies.

B. Non-parametric Techniques Do not assume any knowledge of parameters**.** Use non-parametric techniques to learn a distribution – *e.g. parzen window estimation*

2.1.4.5 Other Point Anomaly Detection Techniques

Other point anomaly detection techniques as illustrated in figure 2.10 includes the following

A. Information Theory Based Techniques

In these techniques, it computes information content in data using information theoretic measures, e.g., entropy, relative entropy, etc. The key idea to these techniques is that the outliers significantly alter the information content in a dataset. Its approach is such that data instances that significantly alter the information content are detected. Also it requires an information theoretic measure. Its advantage is ability to operate in an unsupervised mode. The challenges of information theory based techniques is that it requires an information theoretic measure sensitive enough to detect irregularity induced by very few outliers.

B. Spectral Techniques

This is a technique that its analysis based on Eigen decomposition of data. Key Ideas behind these techniques is to find combination of attributes that Capture bulk of variability and reduced set of attributes can explain normal data well, but not necessarily the outliers. The advantage of Spectral Techniques is that it can operate in an unsupervised mode.

The disadvantages of Spectral Techniques are as follows:

It based on the assumption that anomalies and normal instances are distinguishable in the reduced space, and several methods use Principal Component Analysis of which top few principal components capture variability in normal data, smallest principal component should have constant values and outliers have variability in the smallest component.

C. Visualization Based Techniques

This techniques use visualization tools to observe the data, provide alternate views of data for manual inspection and anomalies are detected visually.

The advantage of visualization based technique is that it keeps a human in the loop.

The disadvantages of visualization based technique are as follows:

- ❖ Works well for low dimensional data
- ❖ Can provide only aggregated or partial views for high dimension data.

2.1.4.6 Contextual Anomaly Detection Technique

Contextual anomaly detection techniques detect context anomalies. The general approach for Contextual Anomaly Detection Techniques are to Identify a context around a data instance (using a set of *contextual attributes*) and Determine if the data instance is anomalous w.r.t. the context (using a set of *behavioral attributes*). And the assumption is that all normal instances within a context will be similar (in terms of behavioral attributes), while the anomalies will be different. The advantage of Contextual Anomaly Detection Techniques is that it Detect anomalies that are hard to detect when analyzed in the global perspective.

The challenges of contextual anomaly detection techniques are as follows:

- ❖ Identifying a set of good contextual attributes
- ❖ Determining a context using the contextual attributes

The contextual anomaly detection Techniques are as follows:

A. Reduction to point outlier detection

It segments data using contextual attributes, and apply a traditional point outlier within each context using behavioral attributes

B. Utilizing structure in data

It builds models from the data using contextual attributes

C. Conditional Anomaly Detection [28]

If a data instance is anomalous in a specific context, then it is termed as a contextual anomaly also referred to as conditional anomaly. The anomalous behavior is determined using the values for the behavioral attributes within a specific context. A data instance might be a contextual anomaly in a given context, but an identical data instance could be considered normal in a different context. This property is key in identifying contextual and behavioral attributes for a contextual anomaly detection technique.

2.1.4.7 Collective Anomaly Detection Techniques

Collective Detection Techniques detects collective anomalies and exploit the relationship among data instances.

A. Sequential anomaly detection

Detects anomalous sequences in a database of sequences, or Detect anomalous subsequence within a sequence, and Data is presented as a set of symbolic sequences such as System call intrusion detection, Proteomics and Climate data.

B. Spatial anomaly detection

Detect anomalous sub-regions within a spatial data set

C. Graph anomaly detection

Detect anomalous sub-graphs in graph data

2.1.4.8 Online Anomaly Detection

The normal behavior is changing through time and it needs to update the "normal behavior" profile dynamically.

Key idea to Online Anomaly Detection Techniques is to update the normal profile with the data records that are "probably" normal, i.e. have very low anomaly score, Time slot i – Data block D_i – model of normal behavior M_i, and Anomaly detection algorithm in time slot $(i+1)$ is based on the profile computed in time slot i.

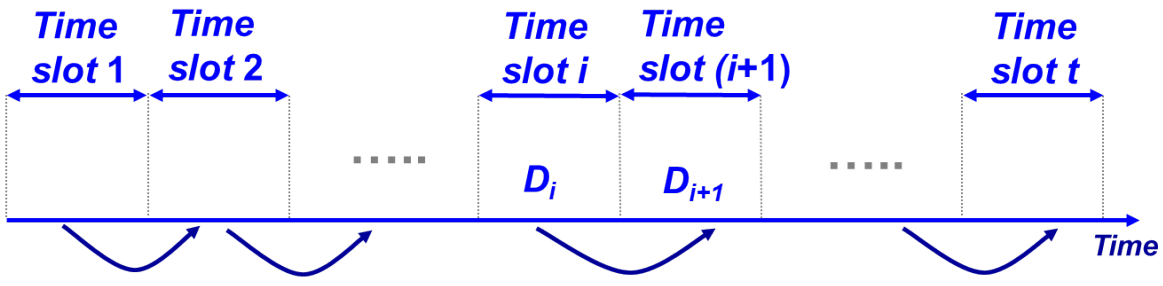


Figure 2.11: Illustrates Online Anomaly Detection Techniques algorithm

2.1.4.9 Distributed Anomaly Detection Techniques

A. Simple data exchange approaches are as follows:

Merging data at a single location, also Exchanging data between distributed locations

B. Distributed nearest neighbouring approaches are as follows:

Exchanging one data record per distance computation, computationally inefficient and also privacy preserving anomaly detection algorithms based on computing distances across the sites

C. Methods based on exchange of models are as follows:

Explore exchange of appropriate statistical / data mining models that characterize normal / anomalous behaviour, identifying modes of normal behaviour, describing these modes with statistical / data mining learning models; and exchanging models across multiple locations and combing them at each location in order to detect global anomalies.

2.2 REVIEW OF EXISTING LITERATURE ON ANOMALY DETECTION

There have been studies or researches in anomaly detection in different problem domains, but in the cloud environment it has not been widely researched on. The Anomaly Detection technique in cloud-based computing is still in view and evolving because it provides challenges that is still in the “cooking pot”. Anomaly detection systems in cloud-based networks detect unwanted traffic in the network and this can be caused by loss of packets, unwanted behavior of application etc.

Patel et al. [29] propose an autonomic agent-based intrusion prevention using the principles of automatic computing. Anomaly based detection technique is used to monitor the system activities and network traffic via autonomous sensors for detection of malicious incidents.

Lee et al. [30] propose a technique that detects suspicious behavior of an intrusion by anomaly level of resource utilization by the user. The main module of the technique is authentication, authorization, and accounting (AAA) component. The anomaly level is based on recent usage history of the user and stored in the database. Intrusion Detection System (IDS) of medium level and low-level security utilize fewer resources so more guest OS can be added without worrying about detection speed. The log files are available for the administrator for auditing. The limitation of this work is that high-level users consume more resources of the system.

Vieira et al. [31] propose an intrusion detection system for grid and cloud computing (GCCIDS). It is a combination of behavior based and knowledge based techniques at middleware layer to detect intrusions. It works in a cooperative manner where each node can detect intrusion and generate alerts for other nodes as well. The limitation of their work is that accurate detection requires more time for training and number of rules is limited.

Dhage et al. [32] propose that an IDS controller should install an IDS instance between cloud service provider (CSP) and user until the user is accessing cloud services. IDS controller collects the log files from all the IDS instances running. The log files help the controller maintain a knowledge-based for all the users based on their activities. It also helps IDS identify the user next time he/she logins. The drawback of knowledge based IDS is that it can only update new samples by the neural network so a workaround is proposed.

Bakshi et al. [33] propose a typical Network Intrusion Detection System (NIDS) for virtualized environments for the detection of DDoS attacks. A NIDS is installed on a virtual switch (through which traffic of all VMs collectively passes). This is analogous to a NIDS being placed at the boundary server in a traditional computing environment. The technique is very much similar to a traditional NIDS. However, the authors have tailored it suitable for the virtualized environment and tested it as such. Limitations of their work are that it can detect known attacks only; no support for large networks, misconfigurations in IDS will miss attacks [30].

Mazzariello et al. [34] have simulated IDS at different locations in cloud to detect DoS attacks on virtual SIP-based hosts. It is a signature based detection technique. The Eucalyptus cloud computing environment has been used for experimentation and SNORT software has been selected as the network IDS. The limitation of their work is that it is unable to detect unknown attacks.

Gupta et al. [36] address main limitations in previous techniques and propose a technique catering for those complexities introduced as a result of inspecting all VMs for all attacks. In their paper, they claim to mitigate this problem by introducing profile-based IDS. They propose an NIDS-based Virtual Machine Introspection (VMI) for the cloud whereby a separate profile is created for each VM based on comparison with known attack signatures and deviation from normal thresholds.

Dastjerdi et al. [36] propose another Distributed Intrusion Detection System (DIDS) based on static and mobile agents. They have modified the Distributed Intrusion Detection Mobile Agents (DIDMA) [37] approach for a DIDS to work in the virtual cloud environments. Dastjerdi et al. mention that their technique reduces network load for less than six VMs in a network. Each mobile agent only analyses a small amount of code. However, if this limit is exceeded, the network loads gets heavier [38, 39]. The limitation of their work is that Limited number of VM can be visited, performance overhead for malicious MA.

Roschke et al. in [40] propose IDS which they deploy on each virtual machine. They mention that this IDS can be either Host-based Intrusion Detection System (HIDS) or Network Intrusion Detection Systems (NIDS) based on the type of sensors deployed and hence the type of data monitored. The drawback of this technique is that it uses multiple IDS sensors and each sensor is deployed on the virtual machine. The limitations are that their work uses multiple sensors and also correlating their data can impact performance

Ibrahim et al. [41] propose a technique that uses VMware specially built Application Programming Interface (APIs) to carry out VM introspection. Their architecture has two main modules Virtual Machine Introspection (VMI) back-end and CloudSec. The authors claim that their technique supports real-time monitoring while bridging the semantic gap. They tested their prototype and found minor overheads. The limitations are as follows: execution suspended in case of event, no defense mechanism, and uses collection of computer software, typically malicious, designed to enable access to a computer or areas of its software that is not otherwise allowed and often masks its existence or the existence of other software only.

Shelke et al. [42] propose a solution for Cross Site Scripting (CSS) and DDoS attacks using a multithreaded network intrusion detection system. The method consists of three modules namely capture module, analysis and processing module, and finally the reporting module. It is a novel approach but the researcher has not provided evidence to prove the concept.

To counter DoS and DDoS attacks in cloud, Lo et al. [43] have proposed and simulated an intrusion detection system. The IDS has four components each playing a specific role. This protects the system from a single point of failure. However, it uses signature-based detection technique due to which unknown attacks are not detected. High computation overhead and inability to detect unknown attacks are their major limitations.

He et al. [44] propose a 3-D IDS architecture. It is a distributed IDS for the users of IaaS in cloud. 3-D IDS is composed of a server and multiple agents. The architecture is a theoretical model and no experimental evidence is presented in the paper. Moreover, the architecture requires the deployment of the server at user end which is not always necessarily deployed at all users.

Siren [45] is another VM-based intrusion detection system. Siren works by detecting malicious software running in the VM that attempt to send information over the network of which the VM is part of. Siren works on the principle that there should be no traffic generated by VM on the network in the absence of human input. If such a state is detected, then Siren flags the traffic as malicious. One of the best features of Siren is its ability to inject crafted human input to investigate for the ad-on malware. The technique seems promising; however the real challenge lies with generating traffic that closely resembles the human input.

Research by Gartner shows an increase in the incidents initiated by application level DDoS flooding attacks [46]; and for their mitigation, access to information in the payload is required. Currently, IDS in cloud are capable enough to detect application layer attack; however, they cannot detect DDoS attacks which use valid packets [47].

Wang, Zhang, and Shin present an algorithm in [48] which will be referred to as SynFinDiff. The core of their algorithm is the CUSUM method described in [49], which belongs to the class of sequential change point detection methods. Roughly speaking, the CUSUM method determines when the mean μ of some independent Gaussian random variables changes to $\mu_1 = \mu_0$. The challenge, then, is to distill network traffic down to a Gaussian random variable with a mean that is stationary during normal operation but changes when a SYN flood attack begins.

Wang et al. use the difference between the count of incoming SYNs and that of outbound FINs, Δ_n, over an observation period of length t_0. The collection period for FINs begins at an offset of t_d later than the start of the collection period for SYNs to allow for the longevity of TCP connections.

Δ_n d is normalized by an exponentially weighted moving average (EWMA) of the number of FINs seen in past observation periods $\bar{F}$.

They define $\bar{X}_n = \Delta n / \bar{F} - a$ - - - - - - - (1)

Where a is a constant chosen to make the mean of $\bar{X}$ negative during normal operation. They then define $y_0 = 0$ and

$y_n = (y_{n-1} + \bar{X}_n)$ + (for n > 0; x is x if x > 0 and 0 otherwise) - - - - (2)

The algorithm reports an attack if $y_n > N$ for some threshold N.

The second algorithm is the CUSUM-based algorithm in [50] by Siris and Papagalou, hereafter termed SynRate. It is similar to SynFinDiff, and in fact Siris and Papagalou explicitly compare their algorithm to that of Wang et al., claiming better performance from their algorithm. The statistic that Siris and Papagalou feed to CUSUM is the number of incoming SYNs seen in a 10-second interval, x_n, minus a EWMA of SYN counts from past intervals, $\bar{\mu}_{n-1}$. The intuition is that the EWMA gives a likely value for the next SYN count, and a significant deviation from that likely value is defined as an anomaly. The final equation derived by Siris and Papagalou for the test statistic g_n is

$$g_n = \left[g_{n-1} + \frac{\alpha \bar{X} n-1}{\sigma^2} \left(x_n - \bar{X}_{n-1} - \frac{\alpha \bar{X} n-1}{2} \right) \right]^+ \quad - \quad - \quad - \quad - \quad (3)$$

Where α is an "amplitude percentage parameter" corresponding to "probable percentage of increase of the mean rate" after an attack begins, and s is the variance of the x. The superscript + indicates that the bracketed expression is forced to 0 if it is less than 0. Similarly to SynFinDiff, SynRate signals an attack when $g_n > h$, for a fixed threshold *h.*

The PCF, or Partial Completion Filter, was introduced by Kompella, Singh, and Varghese in [51]. PCFs are a probabilistic method of detecting significant numbers of SYNs without corresponding FINs (or, generally, significant numbers of the first of any paired operations without the second of the pair). A PCF is built from a set of *k* hash tables, termed "stages" in [51] which use independent hash functions. Each bucket is a counter. When a SYN is seen, for each stage, the pair of the destination IP address and destination port (<dstIP, dstport> for short) is hashed and the corresponding bucket is incremented. Likewise, when a FIN is seen, the same pair is hashed and the corresponding bucket is decremented. If, at any point, all of the buckets to which a <dstIP, dstport> pair hashes are greater than some

threshold, the PCF reports an attack. All of the buckets are zeroed out after a fixed measurement interval.

2.2.1 Summary on the literature Review

In [29, 30, 31 and 32], the characteristics and limitations of host-based IDS were discussed, such that these security challenges can be addressed before a standard technique is recommended for cloud. The ideal performance vs efficiency trade-off in host-IDS has not been achieved so far.

In [33, 34 and 35], the characteristics and limitations of network-based IDS were discussed. The conclusion reached from the analysis is that fast detection is a hurdle in network-based IDS. Furthermore, detection of unknown attacks is an issue since it further degrades the speed of the network.

The characteristics and limitations highlighted in [36, 40, 41, 44, 43, 42 and 45] help us to reach the conclusion that the distributed IDS limit the number of VMs. Furthermore, attention should be provided to reduce computational overhead

CHAPTER THREE
METHODOLOGY

3.1 INTRODUCTION

The application level DDoS flooding attack affects the services in a similar manner as volumetric attack, since they target specific characteristics of an application. This has made it easier for the most appropriate methodology to be considered. Effectively, this chapter will discuss possible directions which this project will take in order to achieve its objectives. The detailed information on the process involved in data acquisition, important attributes of the data, and the information that the attributes gives about the data transfer, also the nature of data transfer are discussed extensively under data analysis section. This research propose Java Logical Program (JLP) SYN Floods Detection algorithm as stated in section 3.6.2, that is capable of solving most of the limitation seen in SynFinDiff, SynRate and PCF algorithms discussed in sections 2.2. Java Logical Program (JLP) algorithm is chosen for this work because of its ability to detect SYN Floods DDoS attack, reduce false positive detection and account for connections closed by RSTs.

3.2 DATA ACQUISITION /ANALYSIS

The aim of the anomaly detection algorithm is to critically examine the effect of Distributed Denial of Service (DDoS) anomaly in network traffic data. The application development is planned as a semi-wider corporate project, aimed at storing captured network traffic traces into database, and performing auto detection of SYN Floods DDoS attacks and displaying result in enhanced form, this will be achieved using JAVA (Object Oriented Programming). It is important to use the analyses of anomaly detection algorithm annotation results in forecasting risks associated with cloud environments at operation time. This research will provide an anomaly based detection algorithm to effectively discover deviations in system and network behaviour. In order to carry out the experiment, Flow records will be used, which are available at WIDE MAWI (Measurement and Analysis on the WIDE Internet) WORKING GROUP. The packet traces to be used in this research work were collected from WIDE MAWI WORKING GROUP repository directory [56] which is publicly made available. The publicly available packet traces was recorded from November 15, 2017 through November 24, 2017.

The MAWI (Measurement and Analysis on the WIDE Internet) Working Group is a working group that has carried out network traffic measurement, analysis, evaluation, and verification from the beginning of the WIDE Project. The WIDE Project carries out research activities through the use of the actual network, but simply operating the network alone does not qualify as research. Its goals are to make use of the knowledge gained through network operations, and to evaluate research results under the actual traffic. In other words, the MAWI (Measurement and Analysis on the WIDE Internet) Working Group evaluate whether the network behaves as it was designed, and learn from unexpected behaviors. Furthermore, if there is a problem, the MAWI (Measurement and Analysis on the WIDE Internet) Working Group seeks out the cause and devises measures to solve the problem. The MAWI Working Group was created to specialize in measurement and evaluation. Additionally, because some form of measurement is often required in any research, the MAWI WG plays a role to share measurement and analysis information across all working groups. The network datasets used in this experiment are 10 in number, captured within 10 Days from November 15th, 2017 to November 24th, 2017 as shown in Table 3.1 respectively. The original files are in pcap extension format. The algorithm should be able to analyse them separately based on their captured date. The total number of network packets samples that will be used in this research experiment is 494,122 Transport Control Protocol (TCP) data.

Table 3.1: Network traffic packet traces captured from Nov 15th, 2017 to Nov 24th, 2017

Row No.	Packet Trace Files	Record Date	Trace Record Start Time	Trace Record Stop Time	TCP Packets
1	15 Nov 2017 – Network Packet	15/11/2017	05:00:00.245111	05:00:01.189483	84,636
2	16 Nov 2017 – Network Packet	16/11/2017	05:00:00.283013	05:00:00.769538	45,977
3	17 Nov 2017 – Network Packet	17/11/2017	05:00:00.312355	05:00:00.723896	32,806
4	18 Nov 2017 – Network Packet	18/11/2017	05:00:00.184597	05:00:00.776104	22,440
5	19 Nov 2017 – Network Packet	19/11/2017	05:00:00.460232	05:00:01.961849	48,163
6	20 Nov 2017 – Network Packet	20/11/2017	05:00:00.234459	05:00:00.836459	52,900
7	21 Nov 2017 – Network Packet	21/11/2017	05:00:00.694923	05:00:01.405467	65,107
8	22 Nov 2017 – Network Packet	22/11/2017	05:00:00.397638	05:00:00.927480	48,423
9	23 Nov 2017 – Network Packet	23/11/2017	05:00:00.417347	05:00:01.186484	48,721
10	24 Nov 2017 – Network Packet	24/11/2017	05:00:00.289857	05:00:00.782565	44,949

3.2.1 Important Attributes of the Datasets

The most important attributes of the datasets are the column header features as shown in Table 3.2

Table 3.2: Column header feature of network traffic packet trace datasets

No.	Time	Source IP	Source Port	Destination Port	Protocol	Length	Acknowledgment	Fin	Info

These column header features of network traffic packet trace datasets are described as follows

1) **No. -** This is a unique number that identifies a connection to a TCP/IP stack. The format is integer.

2) **Time** - is the time since the previous displayed packet was captured.

3) **Source IP** - is the IP (Internet Protocol) address of the device sending the IP packet (the IP unit of data transfer)

4) **Source/Destination Port -** is basically a unique 16 bit integer number given for every socket on your computer with the help of which an external server can deliver requests to each and every process.

5) **Destination IP** - is the IP address of the device to which the packet is being sent.

6) **Network Protocols** - incorporate all the processes, requirements and constraints of initiating and accomplishing communication between computers, servers, routers and other network enabled devices. Network protocols must be confirmed and installed by the sender and receiver to ensure network/data communication and apply to software and hardware nodes that communicate on a network.

7) **Packet Length** - specify the size of the whole packet include header, trailer and the data sent in that packet.

8) **Acknowledgment Number** – is part of the hand-shaking between the two peers. The acknowledgment is sent from the receiver back to the transmitter letting them know the sequence number of the next expected byte.

9) **FIN** - is intended to terminate an established connection. A FIN is greatly preferred when closing down a useful connection as it ensures that the receiver closes down the connection in an orderly fashion and delivers the data in transit.

10) **Info** – provides the summary details on source and destination port, flags, sequence number, window size, Maximum Segment Size (MSS) and connection status of a transmitted packet.

3.2.2 Minimum and Maximum Amount of Packets and Bytes That Can Be Sent Within a Specific Time Frame

The maximum bytes per packet that can be sent within a specific time frame is 8000 byte (64000 bit) per 40 milliseconds. It is not dependent on the kind of network; it is the maximum throughput. And the minimum bytes per packet that can be sent within a specific time frame is 2 byte (16 bit) per 0.01 milliseconds.

3.2.3 TCP Protocol Operation

TCP protocol operations are divided into three phases. Connections must be properly established in a multi-step handshake process (connection establishment) before entering the data transfer phase. After data transmission is completed, the connection termination closes established virtual circuits and releases all allocated resources.

3.2.4 TCP Three Way Handshaking

TCP is stream, connection oriented protocol for packet network Intercommunication, developed by Vinton G. Cerf and Robert K. khan. [52] TCP allows the sending process to deliver data as a stream of bytes and allows the receiving process to obtain data as a stream of bytes. The data/messages are broken by TCP into segments, and each segment consists of a specific format. It uses full duplex service in which data can flow in the both directions. And also uses three way handshaking to establish connection.

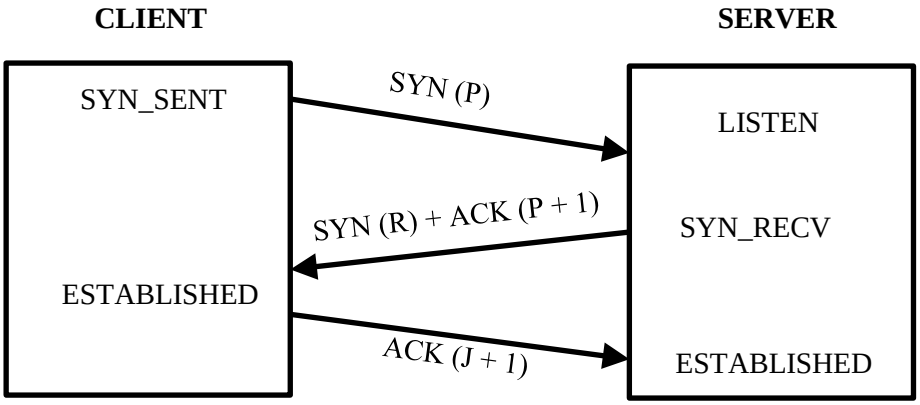


Figure 3.1: TCP 3 Way Handshaking

In Figure 3.1, connection establishment process, firstly the client sends the first segment, a SYN segment to server. After receiving SYN segment from client, server sends a SYN+ACK segment back to client and then client responds with ACK segment and the connection is established.

3.3 APPLICATION DESIGN

Anomaly detection algorithm will also deploy Unified Modelling Language (UML) diagram which is used in object oriented software engineering to model application structures, behaviour and even business processes. Among 14 different types of UML diagram, two will be used in this work Class diagram and Use-case diagram. Data design is described as the phase in which the data expert organizes the main information objects identified during requirements specification into a comprehensive and coherent conceptual data model [53]. Data modelling is a well-established discipline. The most popular conceptual data model is the Entity-Relationship model (ER). The anomaly detection algorithm will deploy Entity-Relationship model in organizing the main information object as described in requirement. The relationship that exists between the tables, which is one of the database objects [that stores information] must also be defined: this is to enable a good and flexible relational database to be developed. Since this application will deploy conceptual data model- the Entity-Relationship (ER), this application must take into consideration the following:

Defining the business process: business process in this context is those duties this application must perform to achieve its objectives such as:

- Uploading packet traces to database functional requirements
- Fetching data from database functional requirements

- Attack detection functional requirement
- Attack detection summary functional requirement
- Attack history functional requirement
- Result display functional requirement
- Export result to CSV(comma delimiter)
- Representation of the detected attacks in bar chart.

Defining the business objects: these are the components that made up the business process. In the context of this application, the objects are as follow:

- Packet trace record object
- Anomalous network packet trace record object

3.4 MODELLING THE DATABASE

This is sketching out the schema, which is actually developing the entity relation diagram (ERD). This Application does not propose yet another data modelling language, but exploits the most successful and popular notation, namely the Entity-Relationship (E-R) model in establishing the objects relationships. This is vital because it makes query creation easy and the joining conditions valid. This application also tends to use Visual Paradigm community edition in achieving the sketching.

3.5 ESTABLISHING RELATIONSHIPS

This is deciding or determining how the database objects relate to each other. The relationship could be one-to-one, one-to-many and many-to-many as showed by Figure 3.2 – 3.4.

Figure 3.2: One-to-one relationship

Figure 3.3: One-to-many relationship

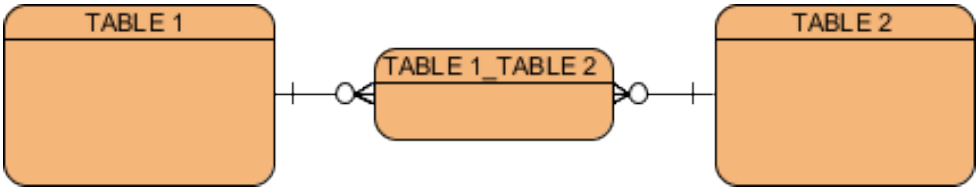


Figure 3.4: Many-to-many relationship

3.6 PROPOSED ARCHITECTURE DESIGN AND ALGORITHM

Architecture design is the definition of the hardware, network, and software components that make up the architecture on which the application delivers its services to users [53]. The inputs of architecture design are the non-functional requirements and the constraints identified during business requirements collection and formalized in the requirement specifications. The output may be any specification that addresses the topology of the architecture in terms of processors, processes, and connections [53]. The Figure 3.5 shows the proposed architecture design for anomaly detection algorithm application.

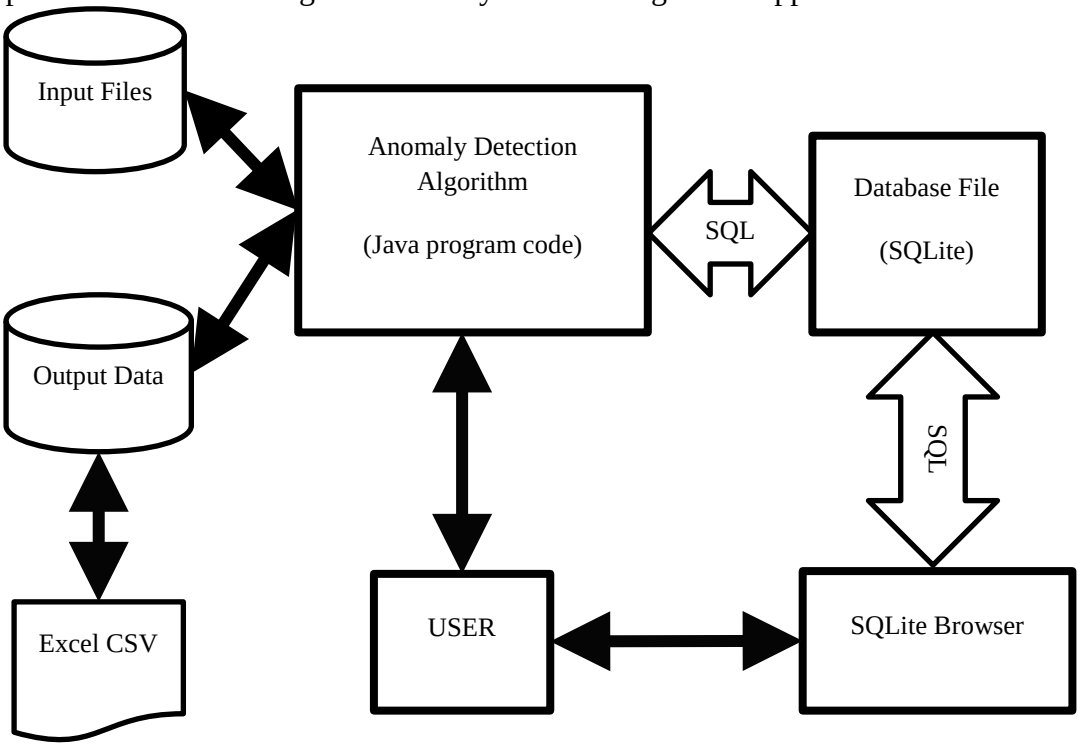


Figure 3.5: proposed architecture design of anomaly detection algorithm application

3.6.1 Criteria for judging the proposed anomaly detection algorithm

This research work is focused on evaluation of the collected data, which are all Transport control protocols (TCP). In judging the proposed anomaly detection algorithm, the judgement will base on TCP three-way handshake in Transmission Control Protocols, which is the method used by TCP to set up a TCP/IP connection over an internet protocol based network. The detected attacks must have the following criteria

Criteria No.1: A SYN floods attack inundate a site with [SYN] segment containing forged (spoofed) IP source addresses with non-existent or unreachable addresses.

Criteria No.2: Host server responds with [SYN, ACK] segments to these addresses and then waits for responding acknowledgement [ACK] segments.

Criteria No.3: Host server will not receive any acknowledgment [ACK] response and eventually time out. This is because in criteria no.2, response was sent to non-existent or unreachable IP addresses.

Criteria No.4: By flooding a host with incomplete TCP connection ([SYN] and [SYN, ACK] without an [ACK]), the detection algorithm result must show that the attacker eventually attempts filling the memory buffer of the victim. Note that once this buffer is full, the host can no longer process new TCP connection requests. The flood might even damage the victim's operating system. Either way, the attack disables the victim and its normal operations.

Criteria No.5: The attacker must send request with multiple IP addresses of the same Port Number against a Victim IP address with varying Port Number or a single Port Number.

Criteria No.6: The attack duration helps to know how harmful the attacks were to the victim. This means that attacks became harmful when it occurs within few minutes and more harmful when it occurs within few seconds.

The proposed anomaly detection algorithm is expected to detect attacks that have the above stated characteristics or criteria.

3.6.2 Proposed Anomaly Detection Algorithm

Applied Algorithm Design is hereby proposed for this work which is also known as "Algorithm Engineering (AE)". Algorithm Engineering is the type of algorithm that focuses on the design, analysis, implementation, optimization, profiling and experimental evaluation of computer algorithms, bridging the gap between algorithm theory and practical applications of algorithms in software engineering.

SYN Floods DDoS Attacks detection algorithm is as follows

Stage 1: Establish database connection

Stage 2: Create packet_traces and attacks table if not existing

Stage 3: Declare all useful variable such as control input variables; Prepare SQL statement and resultset variables.

Stage 4: Remove any existing data on result output table by setting output table model to "0"

Stage 5: Setup TRY block that contains a block of program statements within which an exception might occur, and followed by CATCH block which handles the exception that occurs in associated TRY block.

Stage 6: Authenticate all input variables to make sure there is no empty input field

Stage 7: Create SQL query to delete from attacks table where file_name column is equal to fileName input variable and recordedDate column is equal to input recordedDate. This is with the aim to store new/latest attack record, of the same identifier.

Stage 8: Create SQL query to select all data from packet_traces table where file_name column is equal to filename input variable and info column like [SYN, ACK] and recordedDate column is equal to recordedDate input variable

Stage 9: Create SQL query to select all columns from attacks table and order the selected data by id column in a descending order and limit the selected data to the last

entry. This will help to get last ID column value of attacks table, so that the next stage (Stage 10) will add plus 1 to it and use it as the next ID to store record with.

Stage 10: Create SQL query to insert records gotten from step “8” into attacks table

Stage 11: Create SQL query to select all data from packet_traces table where file_name column is equal to filename input variable and info column LIKE [ACK] and recordedDate column is equal to recordedDate input variable.

Stage 12: Create SQL query to delete all records from attacks table where file_name column is equal to input filename variable AND source_ip column is equal to sourceIP input variable AND source_port column is equal to sourcePort input variable AND destination_ip column is equal to destinationIP input variable AND destination_port column is equal to destinationPort input variable AND recordedDate column is equal to recordedDate input variable AND info column LIKE [ACK] OR source_ip column is equal to sourceIP input variable AND source_port column is equal to sourceIP input variable AND destination_ip column is equal to destinationIP input variable AND destination_port column is equal to destinationPort input variable AND recordedDate column is equal to recordedDate input variable AND info column LIKE [ACK]. This query will help to remove every legitimate (TCP 3 Ways Handshake) network connections requests and leave behind all network connections that do not send final acknowledgement [ACK] response to the host server. And this will be classified as an attack.

Stage 13: Create SQL query to select all records from attacks table where file_name column is equal to fileName input variable and recordedDate column is equal to recordedDate input variable.

Stage 14: Output stage 13 results in the result display table as SYN Floods DDoS attack detected.

Stage 15: Close database connection.

The flow chart diagram for proposed anomaly detection algorithm is shown in Figure 3.6.

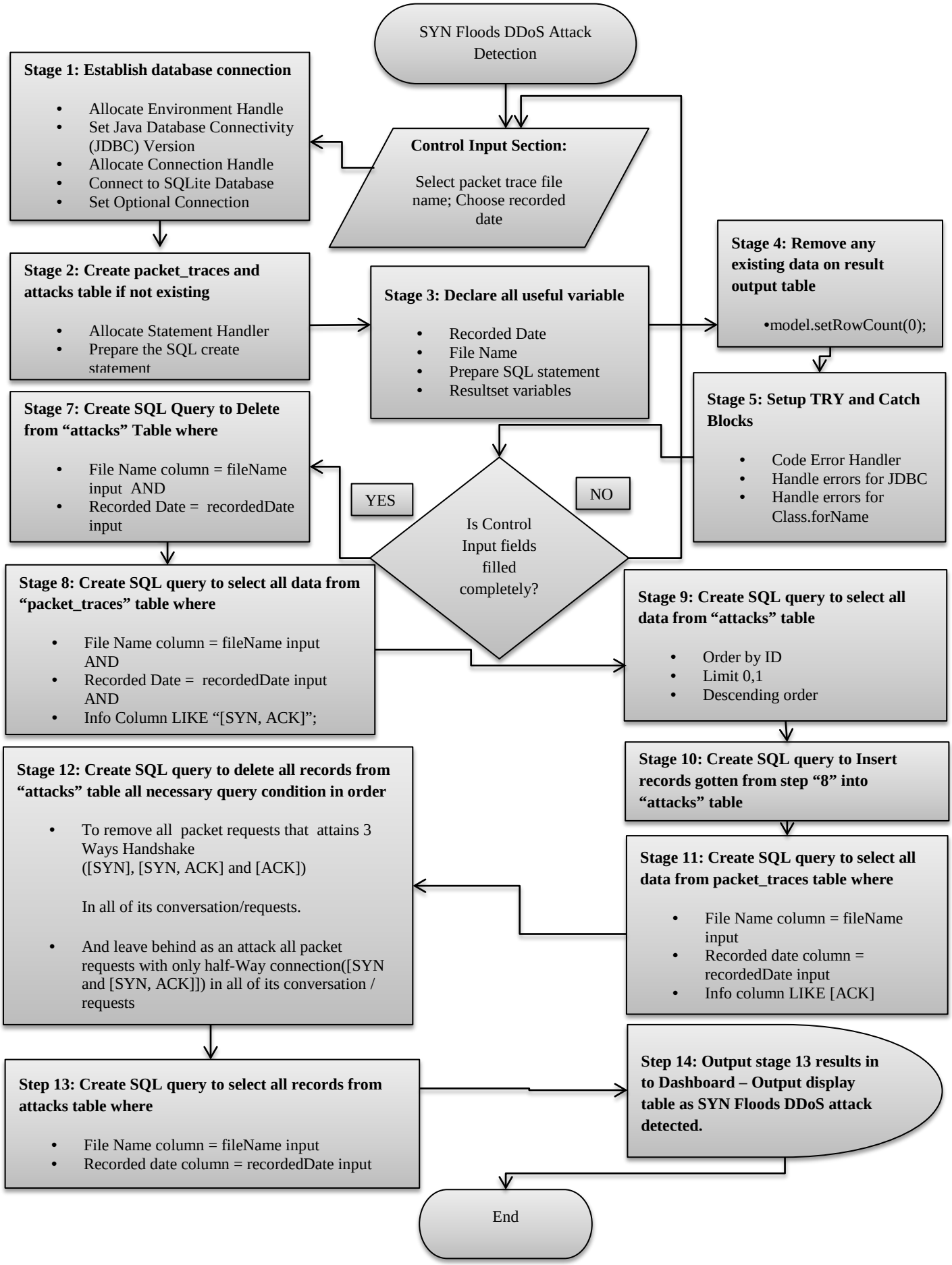


Figure 3.6: Flow chart diagram of anomaly detection algorithm

3.7 SYSTEM REQUIREMENTS SPECIFICATIONS

Anomaly detection algorithm development tools requires operating systems that support the Java VM (Virtual Machine) and has been tested on the platforms listed below

Minimum Hardware Configurations

- Microsoft Windows XP Professional SP3/vista SP1/Window 7 Professional:
    - Processor: 800MHz Intel Pentium III or equivalent
    - Memory: 512
    - Disk space: 750 MB of free disk space.

Recommended Hardware Configurations

- Microsoft Windows XP Professional SP3/vista SP1/Window 7 Professional/Win 8.0/8.1/10.0:
    - Processor: Intel Core i5 or equivalent
    - Memory: 2 GB (32-bit), 4 GB (64-bit)
    - Disk space: 1.5 GB of free disk space

3.8 SOFTWARE REQUIREMENT SPECIFICATION

This section describes anomaly detection algorithm application software to be developed. It lays out functional and non-functional requirements, and includes a set of use cases that describe user interactions that anomaly detection algorithm application must provide.

3.8.1 Programming Language

The programming language of choice for anomaly detection algorithm is Java, which is a general-purpose computer programming language that is concurrent, class-based, object-oriented, and specifically designed to have as few implementation dependencies as possible. It is intended to let application developers "write once, run anywhere" (WORA), meaning that compiled Java code can run on all platforms that support Java without the need for recompilation. No language is simple, but Java considered a much simpler and easy to use object-oriented programming language when compared to the popular programming language, C++. Partially modeled after C++, Java has replaced the complexity of multiple inheritance in C++ with a simple structure called interface, and also has eliminated the use of pointers. Java is designed to make distributed computing easy with the networking capability that is inherently integrated into it. Writing network programs in Java is like sending and

receiving data to and from a file. An interpreter is needed in order to run Java programs. The programs are compiled into Java Virtual Machine code called bytecode. The bytecode is machine independent and is able to run on any machine that has a Java interpreter. Normally, a compiler will translate a high-level language program to machine code and the code is able to only run on the native machine. If the program is run on other machines, the program has to be recompiled on the native machine.

3.8.2 Development Environment

The development environment of choice is NetBeans integrated development environment (IDE). NetBeans is a software development platform written in Java. The NetBeans Platform allows applications to be developed from a set of modular software components called modules. Applications based on the NetBeans Platform, including the NetBeans integrated development environment (IDE), can be extended by third party developers. The NetBeans IDE is primarily intended for development in Java, but also supports other languages, in particular PHP, C/C++ and HTML5. NetBeans is cross-platform and runs on Microsoft Windows, Mac OS X, Linux, Solaris and other platforms supporting a compatible JVM. The NetBeans Team actively supports the product and seeks feature suggestions from the wider community. Every release is preceded by a time for Community testing and feedback.

3.8.3 Database Specification

The database of choice in this project is SQLite. SQLite is a relational database management system contained in a C programming library. In contrast to many other database management systems, SQLite is not a client-server database engine. Rather, it is embedded into the end program. SQLite is ACID-compliant and implements most of the SQL standard, using a dynamically and weakly typed SQL syntax that does not guarantee the domain integrity. SQLite is a popular choice as embedded database software for local/client storage in application software such as web browsers. It is arguably the most widely deployed database engine, as it is used today by several widespread browsers, operating systems, and embedded systems (such as mobile phones), among others.[54] SQLite has bindings to many programming languages.

3.9 IMPLEMENTATION

The NetBeans Integrated Development Environment (IDE) will be used in the implementation of this project. Designing and implementing applications using methods

produce several documents dealing with a larger software development. In such situation, it is paramount to deploy computer-aided software engineering (CASE) tools in the implementation of the application [55]. Anomaly detection algorithm application will run in SQLite embedded database with Java program code.

3.10 SUMMARY

This research work proposed SYN Floods Distributed Denial of Service attacks detection algorithm on Transport Control Protocols (TCP) network traffic packet traces, in order to critically analyze and examine network traffic traces (datasets) and see how this affects detecting anomalies in distributed attacks.

The network traffic packet traces used in this research work were collected from WIDE MAWI WORKING GROUP repository directory, which is publicly made available for researchers. The collected packet traces were recorded from November 15, 2017 through November 24, 2017.

Java programing language was chosen for the development of the proposed algorithm. Java Logical Program (JLP) is the name given to the proposed algorithm.

CHAPTER FOUR

APPLICATION DESIGN / IMPLEMENTATION AND TESTING

This chapter provides detailed information on the processes involved in anomaly detection application design, implementation and testing. The goal was to determine how well anomalies are detected on network traffic traces (datasets) used in the cause of this experiment and to determine how the detection algorithm will affect Distributed Denial of Service attack in TCP packet traces. The result obtained in different timeseries TCP packet traces are compared and used to decide if and how the final results are affected.

4.1 APPLICATION DESIGN

This is actually deploying methodological approach considered in Chapter Three, in defining the requirements to design a database and an outline that describes how the user interface accepts and outputs data, based on the proposed algorithm. The functional requirements of the proposed algorithm are fully explained using Use-Case Diagram, Class Diagram, and Entity Relation diagram. These representations were summarized using the Flow Chart (see figure 4.5).

4.1.1 Use Case Diagram

UML Use Case Diagram shown in Figure 4.1 describes a set of actions (use cases) that anomaly detection algorithm can perform. They are as follows:

- Uploading packet traces(dataset) into SQLite database
- Fetching data from SQLite database
- Initiate SYN Floods Attack detection
- View attack detection summary
- View attack history
- Result display in table rows and
- View bar chart representation of attack detected
- Export result to CSV(comma delimiter) excel document
- Delete detected attack history

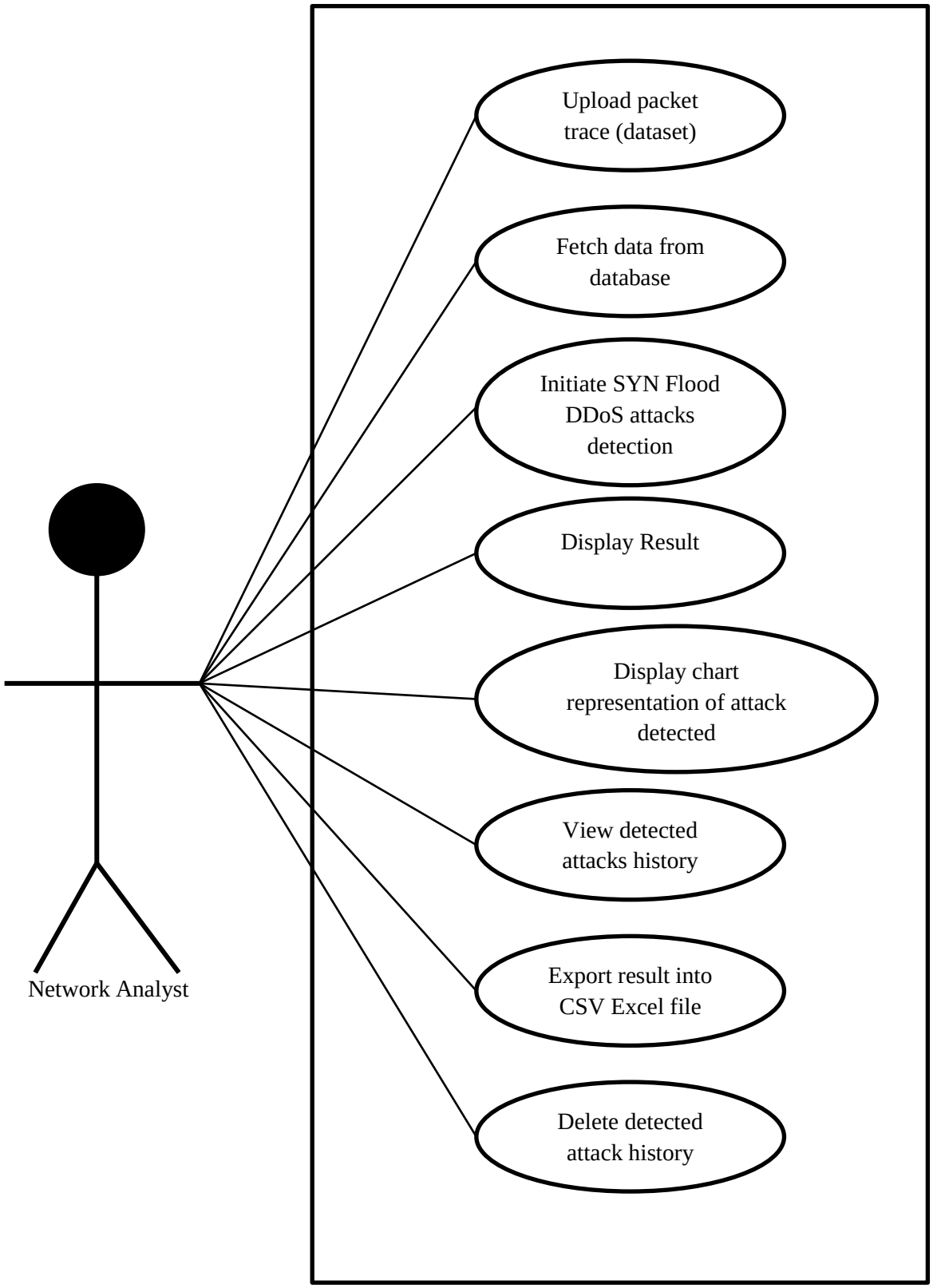


Figure 4.1: Use Case Diagram of anomaly detection algorithm system

4.1.2 Graphical User Interface Design

NetBeans Integrated Development Environment (IDE) was used in designing the graphical user interface of this system. The system is designed such that, it is user friendly and to enable end-user run the anomaly detection algorithm and other functionalities that is shown in use case diagram in Figure 4.1.

Input/output Design Components Used

The inputs and design components that made up the graphical user interphase of this system are as stated in Table 4.1.

Table 4.1: Inputs and design components used for graphical user interphase (GUI)

S/N	Component Name	Palette Content	Description
1.	Panel	Java Swing Container	It is a generic lightweight container
2.	Menu Bar	Java Swing Menus	A container for menu and menu items
3.	Tabbed Pane	Java Swing Container	A component that lets the user switch between a group by clicking on a tab with a given title and/or icon.
4.	Combo Box	Java Swing Control	A component that combines a button or editable field or a drop down list.
5.	JDateChooser	Java Swing Control	It is a calendar java plugin that allows user to choose date by clicking and selecting.
6.	Text Field	Java Swing Control	A lightweight component that allows the editing of a single line of text.
7.	File Chooser	Java Swing Windows	A pane of controls designed to allow user to select a file.
8.	Button	Java Swing Control	A “Push” Button
9.	Menu	Java Swing Menus	A menu for menu items and sub menus
10.	Menu Item	Java Swing Menus	A sing item in a menu

The output and design components that made up this graphical user interface of anomaly detection algorithm system are as stated in Table 4.2.

Table 4.2: Output and design components used for graphical user interphase (GUI)

S/N	Component Name	Palette Content	Description
1.	Table	Java Swing Control	A component that is used to display regular two-dimensional table of cells

4.1.3 Class Diagram

Class diagrams are arguably the most used UML diagram type. It is the main building block of any object oriented solution. It shows the classes in a system, attributes and operations of each class and the relationship between classes. In generating the class diagram of anomaly detection algorithm system, Enterprise Architecture modeling tool was use. Enterprise Architecture modeling tools generated class diagram that has three parts such that the name is placed at the top, attributes in the middle and operations or methods at the bottom. This work provides the class diagram for anomaly detection algorithm as shown in Figure 4.2.

Class diagram in this work shows that the anomaly detection algorithm has four major classes:

1. Dashboard class – it contains most functionalities of this system, which includes all the designed panel, input buttons, labels, text input fields, and menu designs. And also includes their functionality.
2. Dashboard::AnomalyDetection Class – this is the project package class
3. Dashboard::dbconnection class – this is the class where database connection and table creation programme were written.
4. Dashboard::WriteToFileCSV – this is the class that converts SQLite database attacks result into CSV (comma delimiter) Excel format.

The last three java classes listed above are the all class extension of dashboard, having dashboard class as the main class of this application.

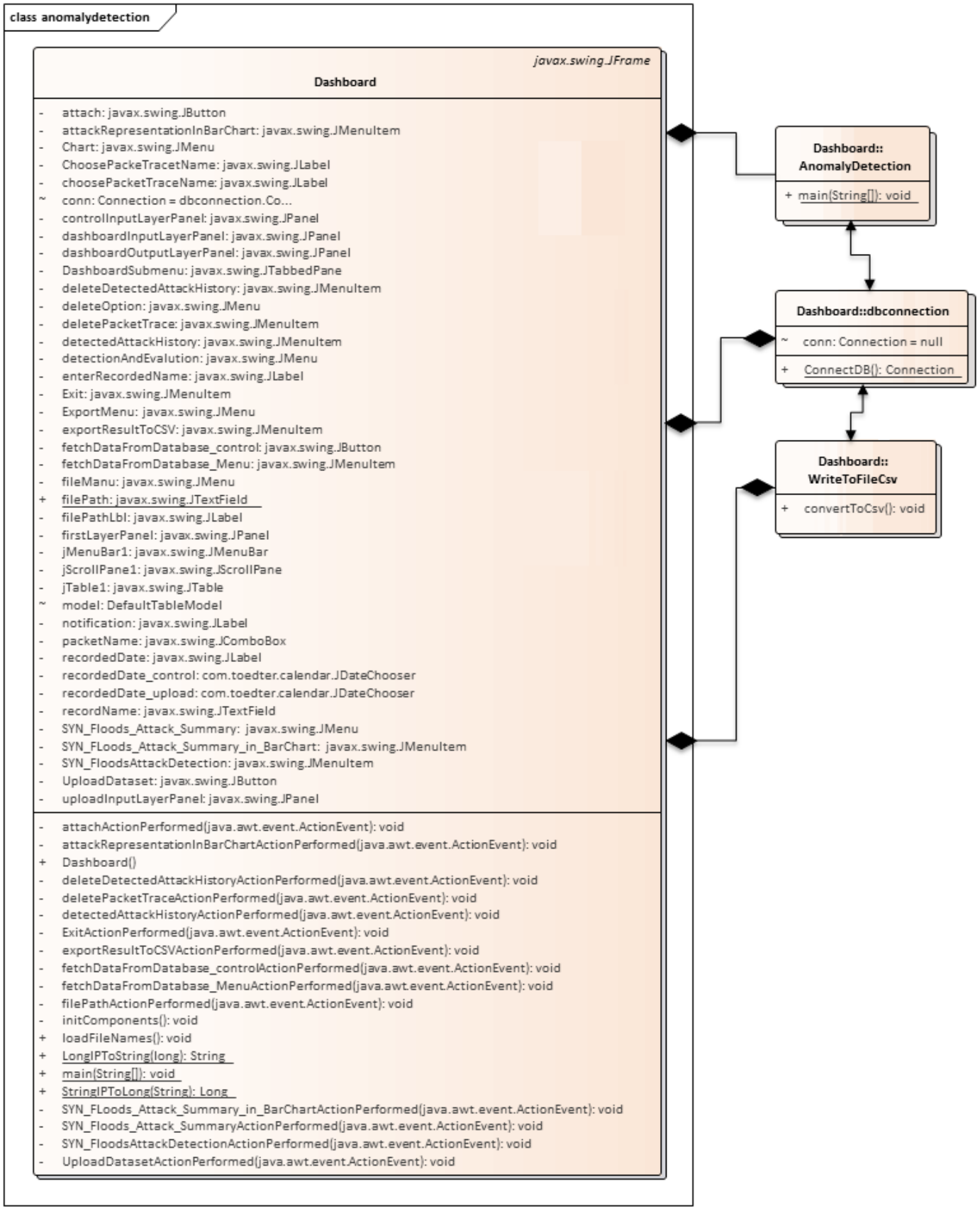


Figure 4.2: Class diagram for anomaly detection algorithm designed application

4.1.4 Designing the database objects

This phase of the application development entails turning those application requirements as described and outlined above into a database design and an outline that describes how the user interface accepts and interprets data. This is also how information has to be organized in the underlying database tables. Given the requirements described for this project, this application needs two database tables in all for effective implementation.

The database table structure design for "PACKET_TRACES" Table is shown in Table 4.3

Table 4.3: Structure design for packet_traces object table

Column ID	*Column name*	*Data type*	*Not Null*	*Primary key*	*Default Value*
0	id	int(11)	Yes	Yes	null
1	no	bigint(100)	No	No	null
2	time	varchar(255)	No	No	null
3	source_ip	bigint(100)	No	No	null
4	source_port	bigint(100)	No	No	null
5	destination_ip	bigint(100)	No	No	null
6	destination_port	bigint(100)	No	No	null
7	protocol	varchar(255)	No	No	null
8	length	bigint(100)	No	No	null
9	arrival_time	text	No	No	null
10	ack	varchar(255)	No	No	null
11	fin	varchar(255)	No	No	null
12	info	varchar(255)	No	No	null
13	file_name	varchar(255)	No	No	null
14	recordedDate	varchar(255)	No	No	null

The database table structure design for "ATTACKS" Table is shown in Table 4.4.

Table 4.4: Structure design for attacks object table

Column ID	*Column name*	*Data type*	*Not Null*	*Primary key*	*Default Value*
0	id	int(11)	Yes	Yes	null
1	no	bigint(100)	No	No	null
2	time	varchar(255)	No	No	null
3	source_ip	bigint(100)	No	No	null
4	source_port	bigint(100)	No	No	null
5	destination_ip	bigint(100)	No	No	null
6	destination_port	bigint(100)	No	No	null
7	protocol	varchar(255)	No	No	null
8	length	bigint(100)	No	No	null
9	arrival_time	text	No	No	null
10	ack	varchar(255)	No	No	null
11	fin	varchar(255)	No	No	null
12	info	varchar(255)	No	No	null
13	file_name	varchar(255)	No	No	null
14	recordedDate	varchar(255)	No	No	null

Defining the application object:

- PACKET_TRACES TABLE: This is database table where all the information / attributes of the network packet traces are uploaded and stored.
- ATTACKS TABLE: This is the database table where the detected SYN Floods attack information is stored.

4.1.5 Database modelling

This is sketching out the schema, which is actually developing the entity relation diagram (ERD). This is vital because it makes query creation easy and the joining conditions valid. Figure 4.3 shows the anomaly detection algorithm database model.

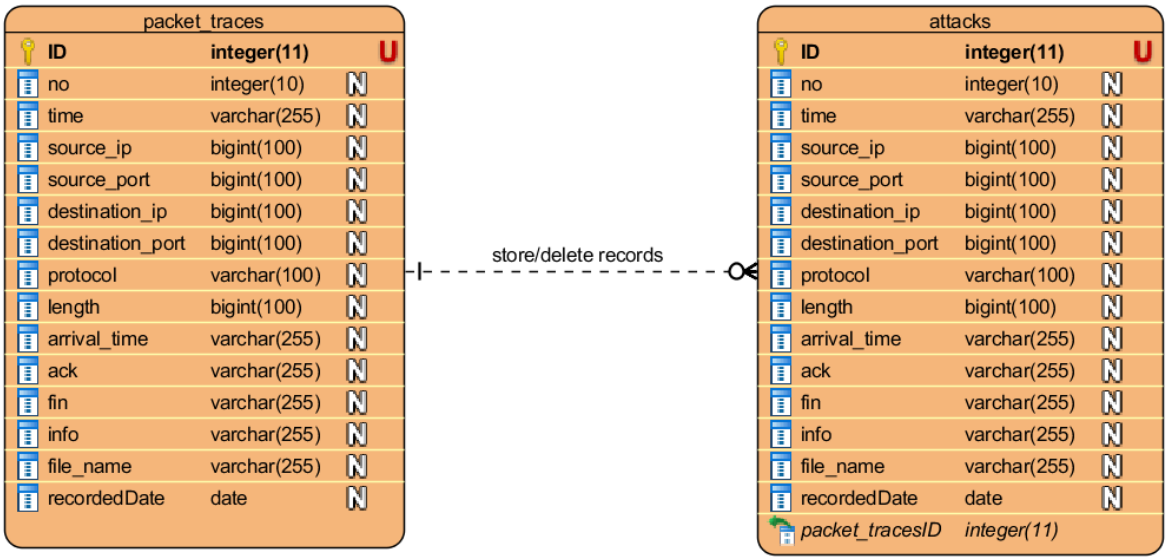


Figure 4.3: Anomaly detection algorithm database table relationship model

4.2 APPLICATION IMPLEMENTATION

In building of this anomaly detection algorithm application and other applications as well, the first thing to do after defining the requirements as done in previous section(4.1), is to successfully create the database objects - tables, constraints, sequences and triggers. It is paramount because, every application is dependent on the tables and other database objects in the schema.

A check on the schema was conducted before creating this application, as to discover errors that may lead to complex problems with the application. So after all these requirements were met, the application was created.

4.2.1 Creating database connection

SQLite Manager was used to create database (db.sqlite) file and used to confirm successful creation of database tables. A database dependent system cannot be run successfully without establishing a successful connection to its database. And so Table 4.5 shows a successful SQLite database (db.sqlite) connection programme writing in Java language which is the programming language of choice in this project.

Table 4.5: Anomaly detection algorithm database table relationship model

Database connection

```
public class dbconnection
{
   Connection conn = null;

   public static Connection ConnectDB()
   {
     Statement stmt = null;

       try
       {
            Class.forName("org.sqlite.JDBC");
            Connection conn = DriverManager.getConnection("jdbc:sqlite:db.sqlite");
            System.out.println("Connected database successfully...");

       }catch(Exception e){
            System.out.println("Exception 1: "+ e);
            return null;
       }
   }
}
```

4.2.2 Creating object in SQLite Manager

SQLite Manager is a multilingual web based tool to manage SQLite database that also enables creation of tables, views, triggers, functions, insert, update, delete records and management of Trigger. For table creation, once in SQLite manager environment, the wizard takes you through the four phases of creating table. The first phase was the column phase- here column name, data type, precision, scale and null condition are assigned. The second phase was assignment of primary keys. The third was assignment of foreign keys and referencing the corresponding tables and columns as well. The fourth phase is adding constraints, in this context the added constraint is the “check constraint”. Here data type and primary keys of referencing tables must match for the table to be created if not, error code will be generated. Here creation of database table was done using java programme.

Program Section Creating “packet_traces” and “attacks” database tables are shown in Table 4.6 and 4.7 respectively.

Table 4.6: Creation of packet_taces database table

Creating packet_traces table

```
try{
stmt = conn.createStatement();
        String packet_traces = "CREATE TABLE IF NOT EXISTS packet_traces"+
                                "(id int primary key not null, " +
                                "no bigint(100), " +
                                "time VARCHAR(255), " +
                                "source_ip bigint(100), " +
                                "source_port bigint(100), " +
                                "destination_ip bigint(100), " +
                                "destination_port bigint(100), " +
                                "protocol VARCHAR(255), " +
                                "length bigint(100), " +
                                "arrival_time TEXT, " +
                                "ack VARCHAR(255), " +
                                "fin VARCHAR(255), " +
                                "info TEXT, " +
                                "file_name VARCHAR(255), "+
                                "recordedDate date)";
              stmt.executeUpdate(packet_traces);
       System.out.println("packet_traces table created successfully...");
 }catch(Exception e){ System.out.println("Exception 1: "+ e); return null;}
```

Table 4.7: Creation of attacks database table

Creating attacks table

```
try{
       String attacks = "CREATE TABLE IF NOT EXISTS attacks"+
                                "(id int primary key not null, " +
                                "no bigint(100), " +
                                "time VARCHAR(255), " +
                                "source_ip bigint(100), " +
                                "source_port bigint(100), " +
                                "destination_ip bigint(100), " +
                                "destination_port bigint(100), " +
                                "protocol VARCHAR(255), " +
                                "length bigint(100), " +
                                "arrival_time TEXT, " +
                                "ack VARCHAR(255), " +
                                "fin VARCHAR(255), " +
                                "info TEXT, " +
                                "file_name TEXT, " +
                                "recordedDate date)";
              stmt.executeUpdate(attacks);
```

```
            System.out.println("attacks table created successfully...");
                return conn;
    }catch(Exception e){ System.out.println("Exception 1: "+ e); return null;}
```

4.2.3 Graphical User Interface (GUI) Implementation

The first step to take after creating the database objects that support the anomaly detection algorithm application is the creation of user interface. Here, all the input design components stated in section 4.1.2 were used to implement the GUI for anomaly detection algorithm application and also it was designed in accordance to the Use-Case Diagram and class diagram (dashboard) attributes functions as the major part of application design requirements. Figure 4.4 shows the successful design implementation of graphical user interface of anomaly detection algorithm application.

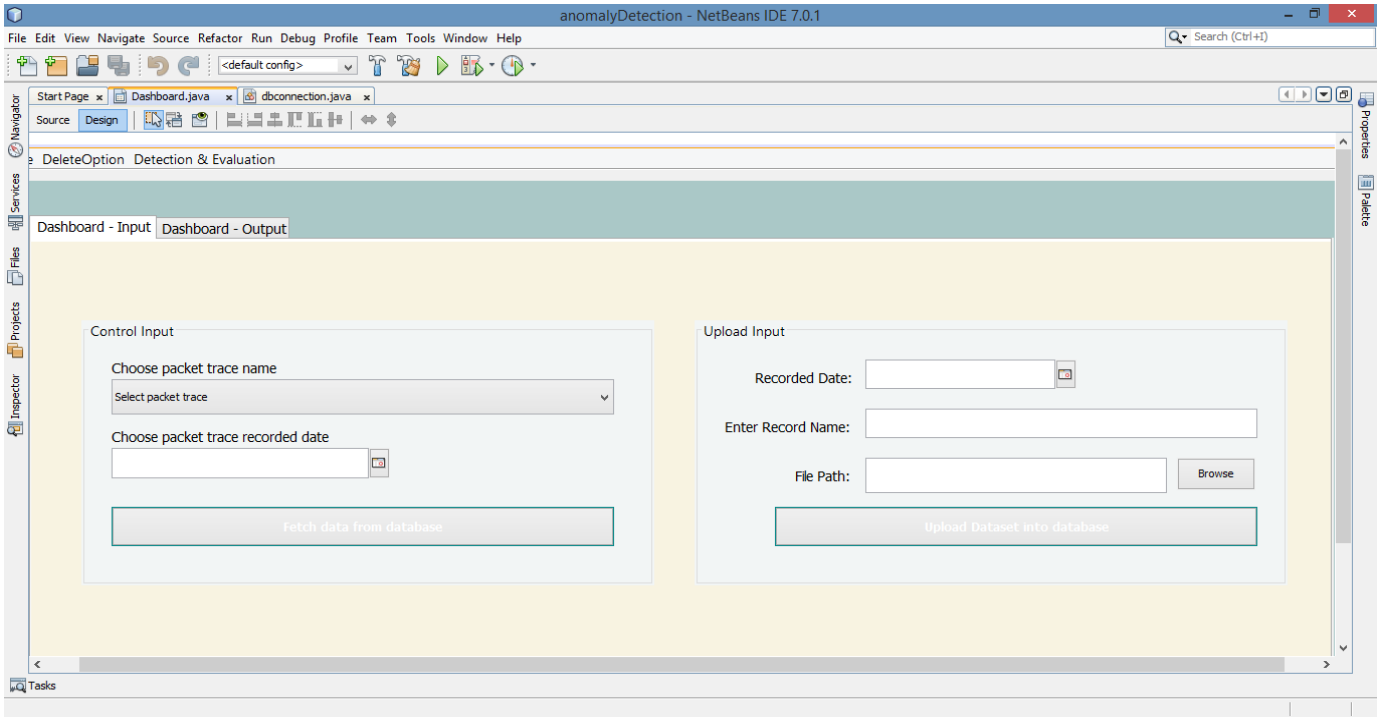


Figure 4.4: Graphical User Interface of anomaly detection algorithm design

The input design phase is classified into two major parts Control Input and Upload Input respectively.

Control Input – is the input section that its parameters are used as the key condition for querying SQL database. It determines the output/result of SQL query initiated. These parameters functions include selecting the network packet trace name on which analysis will be performed, second parameter is used to select the date in which the selected packet trace name file was recorded or captured.

Upload Input – is the input section that its parameter are used to specify the nature of the network traffic packet trace that is to be uploaded into the SQLite database packet_traces. The first parameter specifies the date at which the network packet trace file was recorded. The second parameter demands that the user should enter the name of the network packet trace dataset that is to be uploaded.

The third parameter demands that the user should browse and select the packet trace dataset to be uploaded.

The flow chart diagram for the uploading process of network packet trace data into database is as shown in Figure 4.5. It describes the functional java code that made up the uploading process of the network packet trace.

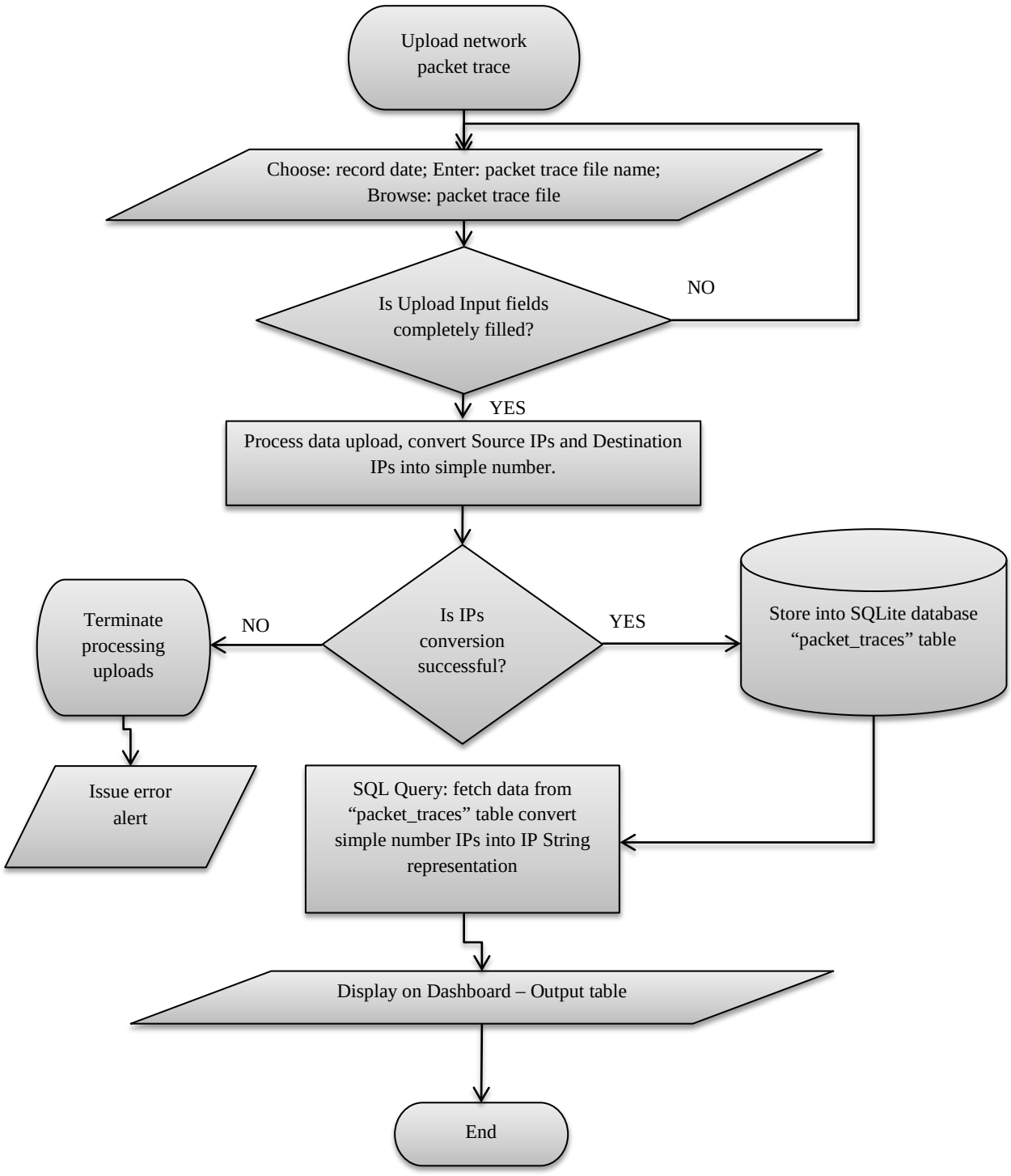


Figure 4.5: Flow chart diagram for the uploading process of network packet trace data into database

The menu section of this system is classified into three major parts File, DeleteOption and Detection & Evaluation respectively. These parts are described respectively as follows:

4.2.4 File Menu

Its submenu includes 'Fetch data' from database, Exporting attacks detected result to CSV (comma delimiter) and 'Exit' as part of the functional requirements of anomaly detection algorithm system.

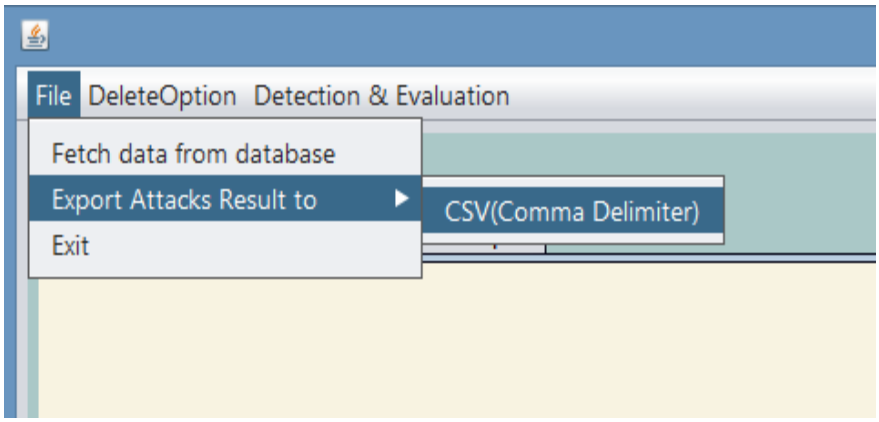


Figure 4.6: File menu implemented functionalities

A. Fetch Data from Database – as the name implies, this submenu works with the control input parameters in order to perform its required action. Here the user fills the control input parameters completely before clicking on "Fetch data from database" submenu, else error message alert will be displayed. Its function is to load the stored network packet trace into the output table as is illustrated in Figure 4.7. The java code written for adding functionality to this submenu is shown in Appendix A

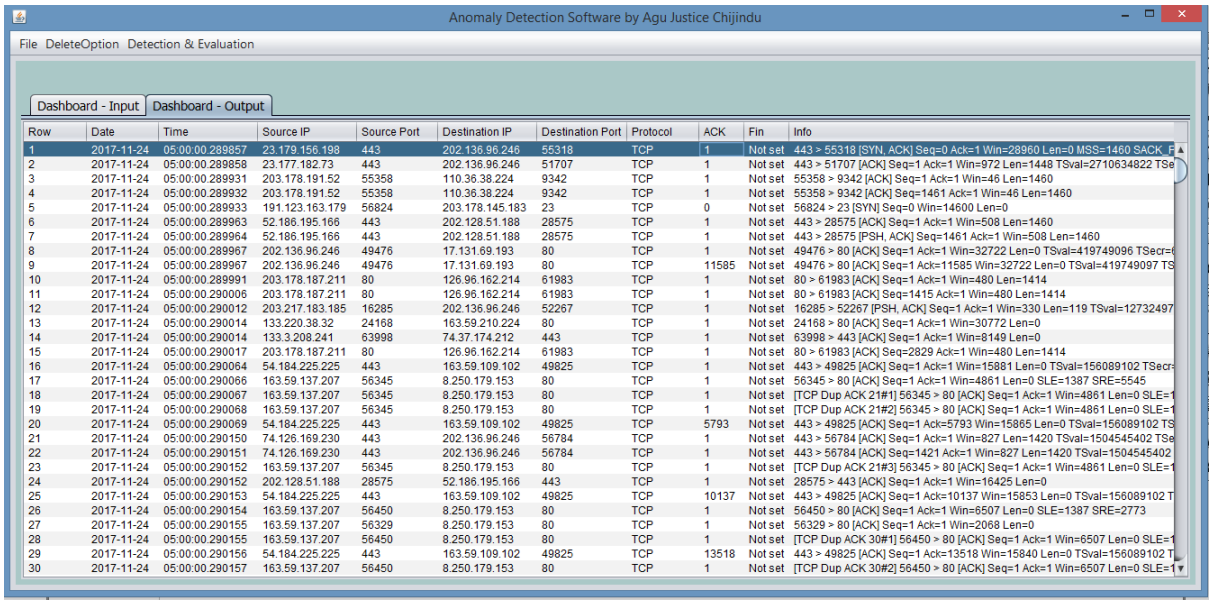
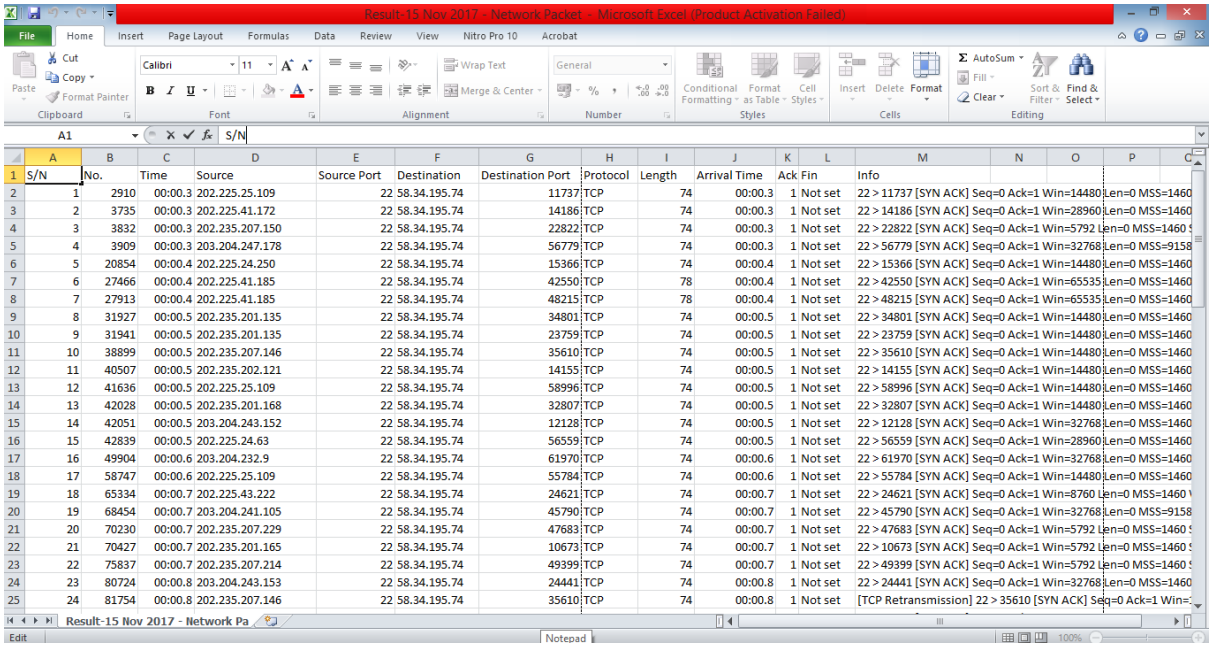


Figure 4.7: Illustrates fetch data from database submenu

B. Exporting attacks result to CSV

This submenu converts to CSV (comma delimiter) excel file; the attacks detected in a packet trace data stored in attack table. This gives option for further documentation or examination of the exported data where necessary. Figure 4.8 shows successfully exported attack file in CSV excel format. Code for adding functionality to this submenu is shown in Appendix B.



S/N	No.	Time	Source	Source Port	Destination	Destination Port	Protocol	Length	Arrival Time	Ack	Fin	Info
1	2910	00:00.3	202.225.25.109	22	58.34.195.74	11737	TCP	74	00:00.3	1	Not set	22 > 11737 [SYN ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1460
2	3735	00:00.3	202.225.41.172	22	58.34.195.74	14186	TCP	74	00:00.3	1	Not set	22 > 14186 [SYN ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460
3	3832	00:00.3	202.235.207.150	22	58.34.195.74	22822	TCP	74	00:00.3	1	Not set	22 > 22822 [SYN ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1460
4	3909	00:00.3	203.204.247.178	22	58.34.195.74	56779	TCP	74	00:00.3	1	Not set	22 > 56779 [SYN ACK] Seq=0 Ack=1 Win=32768 Len=0 MSS=9158
5	20854	00:00.4	202.225.24.250	22	58.34.195.74	15366	TCP	74	00:00.4	1	Not set	22 > 15366 [SYN ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1460
6	27466	00:00.4	202.225.41.185	22	58.34.195.74	42550	TCP	78	00:00.4	1	Not set	22 > 42550 [SYN ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460
7	27913	00:00.4	202.225.41.185	22	58.34.195.74	48215	TCP	78	00:00.4	1	Not set	22 > 48215 [SYN ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460
8	31927	00:00.5	202.235.201.135	22	58.34.195.74	34801	TCP	74	00:00.5	1	Not set	22 > 34801 [SYN ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1460
9	31941	00:00.5	202.235.201.135	22	58.34.195.74	23759	TCP	74	00:00.5	1	Not set	22 > 23759 [SYN ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1460
10	38899	00:00.5	202.235.207.146	22	58.34.195.74	35610	TCP	74	00:00.5	1	Not set	22 > 35610 [SYN ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1460
11	40507	00:00.5	202.235.202.121	22	58.34.195.74	14155	TCP	74	00:00.5	1	Not set	22 > 14155 [SYN ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1460
12	41636	00:00.5	202.225.25.109	22	58.34.195.74	58996	TCP	74	00:00.5	1	Not set	22 > 58996 [SYN ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1460
13	42028	00:00.5	202.235.201.168	22	58.34.195.74	32807	TCP	74	00:00.5	1	Not set	22 > 32807 [SYN ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1460
14	42051	00:00.5	203.204.243.152	22	58.34.195.74	12128	TCP	74	00:00.5	1	Not set	22 > 12128 [SYN ACK] Seq=0 Ack=1 Win=32768 Len=0 MSS=1460
15	42839	00:00.5	202.225.24.63	22	58.34.195.74	56559	TCP	74	00:00.5	1	Not set	22 > 56559 [SYN ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460
16	49904	00:00.6	203.204.232.9	22	58.34.195.74	61970	TCP	74	00:00.6	1	Not set	22 > 61970 [SYN ACK] Seq=0 Ack=1 Win=32768 Len=0 MSS=1460
17	58747	00:00.6	202.225.25.109	22	58.34.195.74	55784	TCP	74	00:00.6	1	Not set	22 > 55784 [SYN ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1460
18	65334	00:00.7	202.225.43.222	22	58.34.195.74	24621	TCP	74	00:00.7	1	Not set	22 > 24621 [SYN ACK] Seq=0 Ack=1 Win=8760 Len=0 MSS=1460
19	68454	00:00.7	203.204.241.105	22	58.34.195.74	45790	TCP	74	00:00.7	1	Not set	22 > 45790 [SYN ACK] Seq=0 Ack=1 Win=32768 Len=0 MSS=9158
20	70230	00:00.7	202.235.207.229	22	58.34.195.74	47683	TCP	74	00:00.7	1	Not set	22 > 47683 [SYN ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1460
21	70427	00:00.7	202.235.201.165	22	58.34.195.74	10673	TCP	74	00:00.7	1	Not set	22 > 10673 [SYN ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1460
22	75837	00:00.7	202.235.207.214	22	58.34.195.74	49399	TCP	74	00:00.7	1	Not set	22 > 49399 [SYN ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1460
23	80724	00:00.8	203.204.243.153	22	58.34.195.74	24441	TCP	74	00:00.8	1	Not set	22 > 24441 [SYN ACK] Seq=0 Ack=1 Win=32768 Len=0 MSS=1460
24	81754	00:00.8	202.235.207.146	22	58.34.195.74	35610	TCP	74	00:00.8	1	Not set	[TCP Retransmission] 22 > 35610 [SYN ACK] Seq=0 Ack=1 Win=

Figure 4.8: Successful exported attack file in csv excel format

C. Exit

The Exit submenu enables user to close opened application window. Table 4.8 shows the java code for adding functionality to exit submenu.

Table 4.8: Functionality code for exit submenu

Exit functionality
java.lang.System.exit(0);

4.2.5 Delete Option

This menu provides feature that will enable the system user to delete a packet trace record from database. Successful deleting of any packet trace database record is dependent on the control input parameters. The user has to select packet name to be deleted and choose the date the packet trace was captured before clicking on “delete a packet trace from database” submenu. Also, DeleteOption menu provides the user option to “delete all attacks history”

when necessary, and this does not require any of the input parameters. Figure 4.9 illustrates DeleteOption menu and also Appendices C and D show java code for adding functionality to DeleteOption submenus.

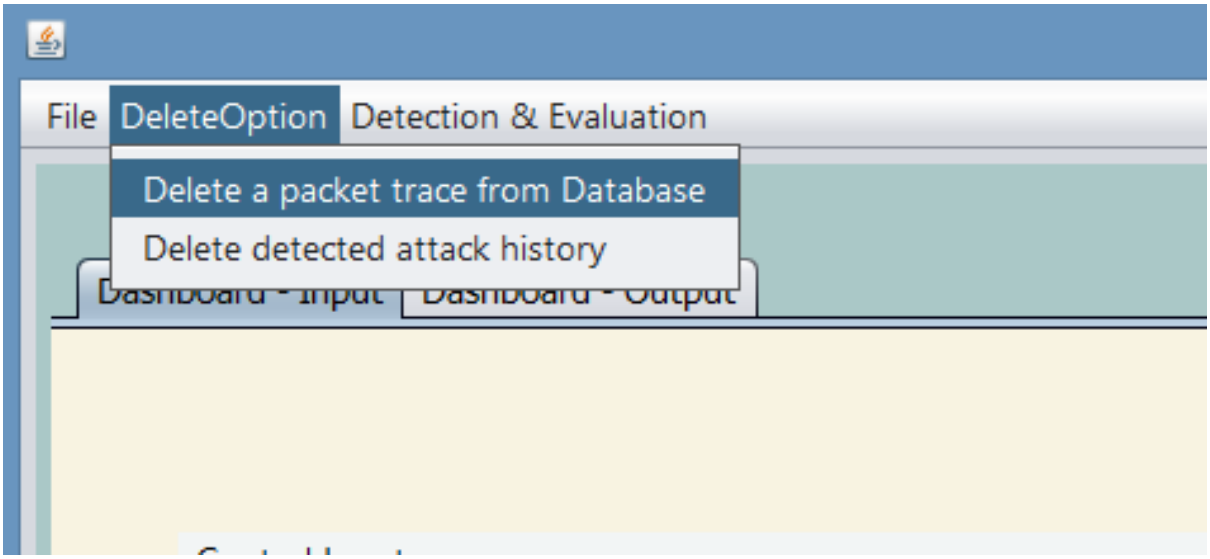


Figure 4.9: Delete Option system functionality

4.2.6 Detection and Evolution

In Figure 4.10 detection and evaluation menu contains the following functionalities

A. Detection of SYN Floods DDoS attack – this is where the proposed anomaly detection algorithm for network traffic packet is applied. The performance of this SYN Floods detection submenu depends on the control input parameter.

Firstly, the control input parameters are defined by selecting the packet trace name and recorded date for the trace and submitted for the execution of SYN Floods detection task. The proposed algorithm looks into database packet_traces table record and fetches records where the selected control input parameters and INFO column is [SYN, ACK]. If matches were found, then the next lines of code will store all match results into attacks table. Subsequently, record deleting task from attacks-table will be performed. It will delete any row/record where [ACK] is found inside INFO column using Source IP, Source Port, Destination IP and Destination Port. Or delete any row/record where [ACK] is found inside INFO column using Destination IP and Destination Port, Destination IP and Destination Port columns matching with their respective column associated values fetched from packet_traces table where [ACK] exists in INFO column. Then use these records to delete from attacks table record where this

value exists. These deleted records are those network packets that have TCP 3 Ways Hand Shake connection. In other words, the deleted records are the legitimate TCP packets connections. At this stage the remaining packet records in the attacks database table are now seen as SYN Floods Distributed Denial of Service attack detected. The detected attack results are displayed on the output table. The functionality code that implements the proposed anomaly detection algorithm is shown in Appendix E.

B. Detected Attack History – this is another submenu under DDoS attack submenu as it is shown in Figure 4.10. Its function is to display on the dashboard – Output section table, the already detected SYN Floods attacks existing in attacks database table, with respect to the specified control input parameters for a particular network packet trace record. Program section of "Detected Attack History" submenu was added as is shown in Appendix F.

C. Chart Representation of SYN Floods Attacks Detected in Bar Chart – this submenu, when clicked, makes use of the selected control input parameter, such as the selection of packet trace name and the captured date of the trace selected to perform data analysis and represent the number of attacks detected in bar chart. The program code for this is shown in Appendix G.

D. SYN Floods Attacks Summary – this submenu provides the summary of all SYN Floods attacks detected by grouping the attacks based on the each packet traces and captured date of each network packet traces. Its program is shown in Appendix H.

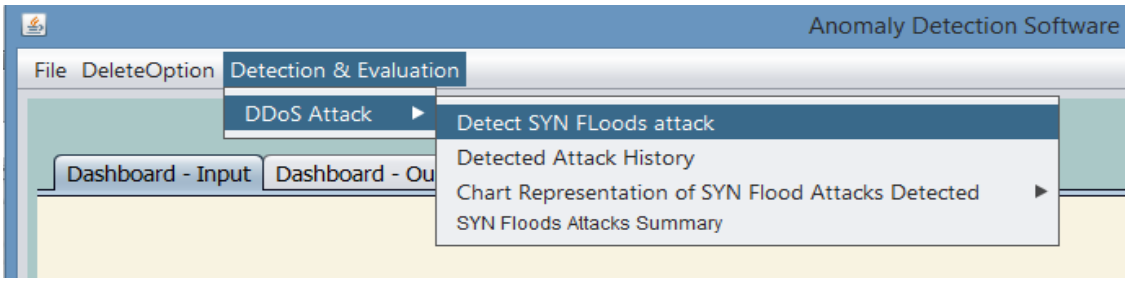


Figure 4.10: Detection and Evaluation system functionalities

4.3 APPLICATION TESTING

Anomaly Detection Algorithm application was executed without errors. It could only generate error where wrong datatype input, authentication or authorization is made. There was no error messages from the netbeans integrated development environment and that means that the application runs but it does not prove its functionality and interoperability. To ascertain that an application is functional- that it meets its aims and objectives, a series of test

should be conducted. Figure 4.11 shows successful testing of the anomaly detection algorithm application dashboard.

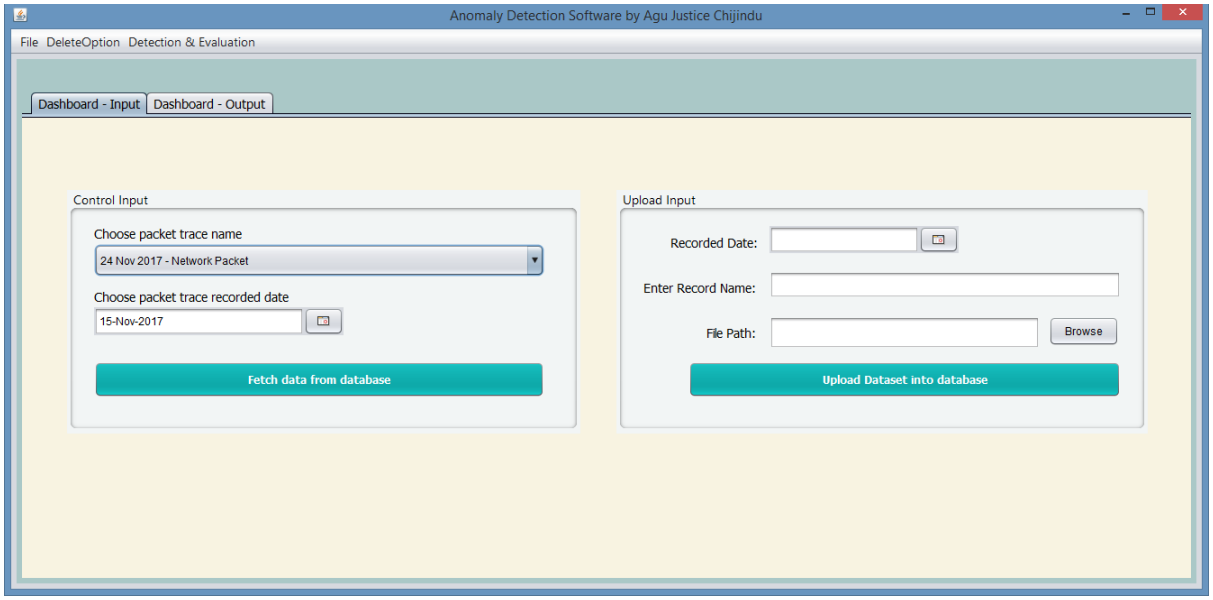


Figure 4.11: Testing of anomaly detection dashboard

4.3.1 Testing the Packet Trace file Upload

In order to test the packet trace file uploading functionality, Flow records were used, which were collected from WIDE MAWI WORKING GROUP repository [56] directory (which is publicly made available). The publicly available packet traces were recorded from November 15, 2017 through November 24, 2017.

The MAWI (Measurement and Analysis on the WIDE Internet) Working Group is a working group that has carried out network traffic measurement, analysis, evaluation, and verification from the beginning of the WIDE Project. The WIDE Project carries out research activities through the use of the actual network, but simply operating the network alone does not qualify as research. Its goals are to make use of the knowledge gained through network operations, and to evaluate research results under the actual traffic. In other words, the MAWI (Measurement and Analysis on the WIDE Internet) Working Group evaluate whether the network behaves as it was designed, and learn from unexpected behaviors.

The network datasets used in this experiment are 10 in number, captured within 10 Days from November 15th, 2017 to November 24th, 2017 as shown in Table 3.1 respectively. The original files are in pcap extension format. The algorithm was able to analyse them separately based on their captured date. The total number of network packets samples that will be used in this research experiment is 494,122 Transport Control Protocol (TCP) data.

Packet trace uploading section provides the user the option to upload all packet trace files into the database management system. For packet trace to be uploaded, the user has to fill in the upload input parameter form completely, else an error message alert will be prompted. The upload section was implemented and tested as it shown in figure 4.12 and 4.13. The uploaded data are displayed on the dashboard – output table and task completed alert prompts. The packet trace file to be uploaded has the following: no, Time, Source IP, Source Port, Destination IP, Destination Port, Protocols, Arrival Time, Length, Acknowledgement, Finish and Info column as the data inputs to the upload section. All these columns are stored into the database packet_trace table, from which every other analysis is performed.

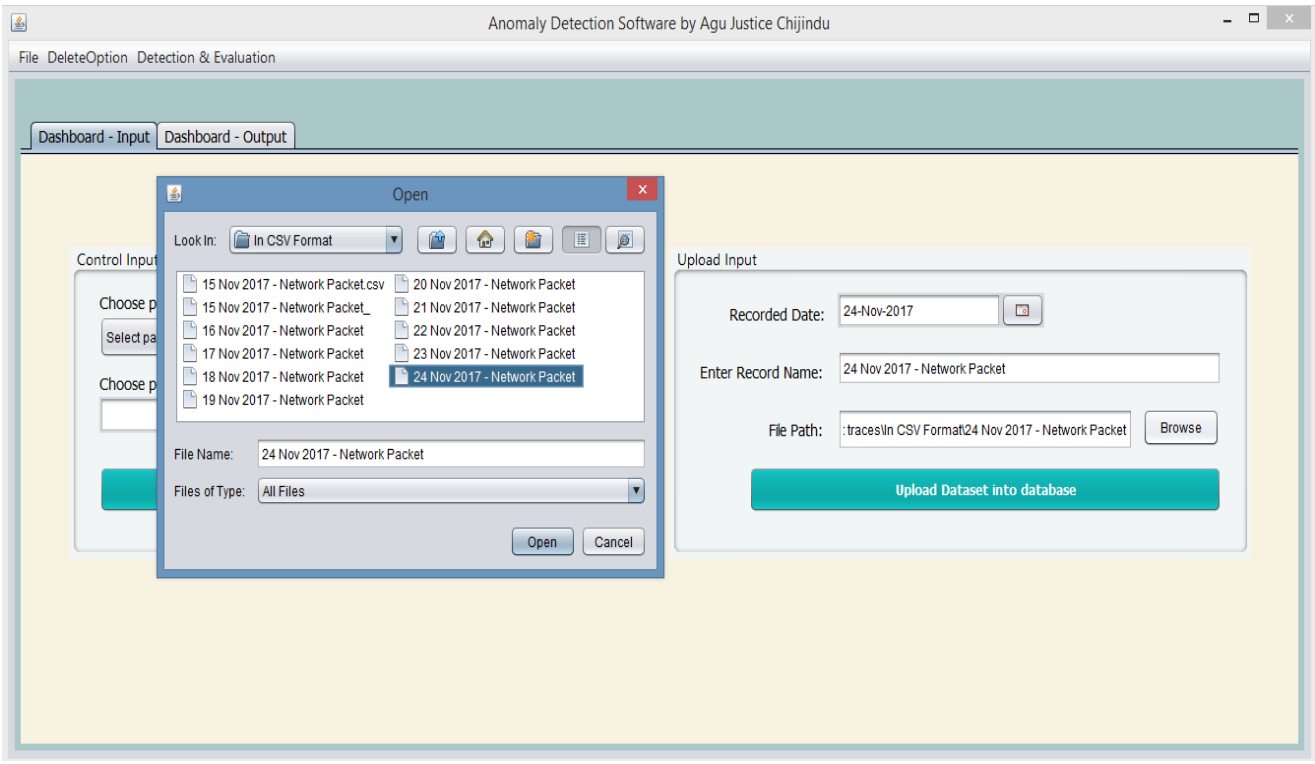
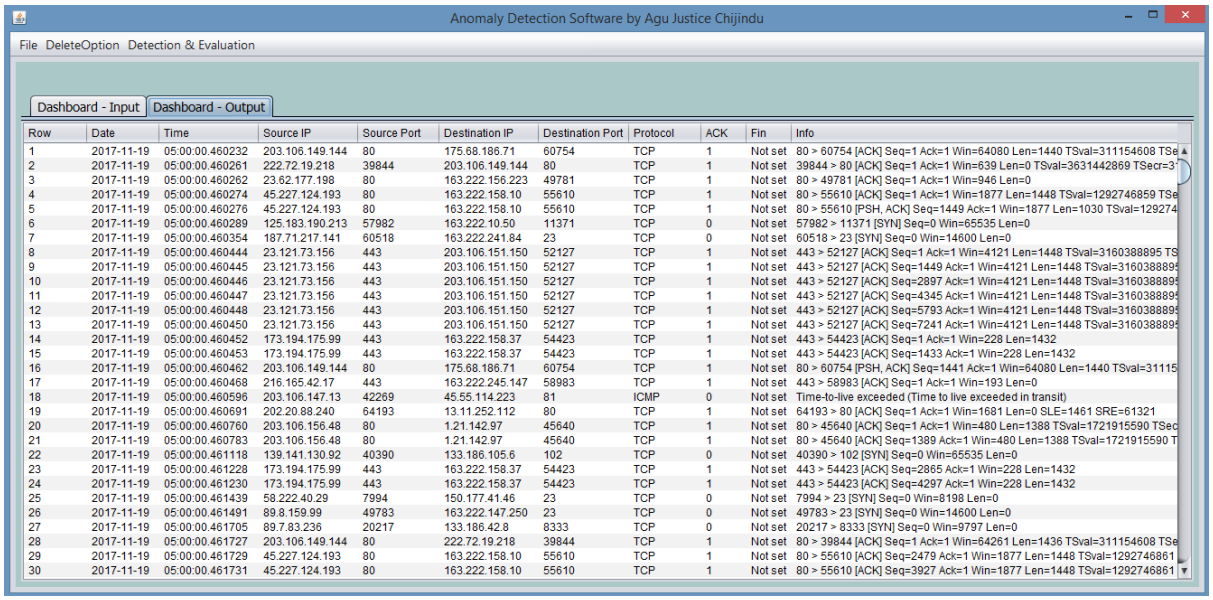


Figure 4.12: Packet traces file uploading process testing



Row	Date	Time	Source IP	Source Port	Destination IP	Destination Port	Protocol	ACK	Fin	Info
1	2017-11-19	05:00:00.460232	203.106.149.144	80	175.68.186.71	60754	TCP	1	Not set	80 > 60754 [ACK] Seq=1 Ack=1 Win=64080 Len=1440 TSval=311154608 TSe
2	2017-11-19	05:00:00.460261	222.72.19.218	39844	203.106.149.144	80	TCP	1	Not set	39844 > 80 [ACK] Seq=1 Ack=1 Win=639 Len=0 TSval=3631442869 TSecr=3
3	2017-11-19	05:00:00.460262	23.62.177.198	80	163.222.156.223	49781	TCP	1	Not set	80 > 49781 [ACK] Seq=1 Ack=1 Win=946 Len=0
4	2017-11-19	05:00:00.460274	45.227.124.193	80	163.222.158.10	55610	TCP	1	Not set	80 > 55610 [ACK] Seq=1 Ack=1 Win=1877 Len=1448 TSval=1292746859 TSe
5	2017-11-19	05:00:00.460276	45.227.124.193	80	163.222.158.10	55610	TCP	1	Not set	80 > 55610 [PSH, ACK] Seq=1449 Ack=1 Win=1877 Len=1030 TSval=129274
6	2017-11-19	05:00:00.460289	125.183.190.213	57982	163.222.10.50	11371	TCP	0	Not set	57982 > 11371 [SYN] Seq=0 Win=65535 Len=0
7	2017-11-19	05:00:00.460354	187.71.217.141	60518	163.222.241.84	23	TCP	0	Not set	60518 > 23 [SYN] Seq=0 Win=14600 Len=0
8	2017-11-19	05:00:00.460444	23.121.73.156	443	203.106.151.150	52127	TCP	1	Not set	443 > 52127 [ACK] Seq=1 Ack=1 Win=4121 Len=1448 TSval=3160388895 TS
9	2017-11-19	05:00:00.460445	23.121.73.156	443	203.106.151.150	52127	TCP	1	Not set	443 > 52127 [ACK] Seq=1449 Ack=1 Win=4121 Len=1448 TSval=3160388895
10	2017-11-19	05:00:00.460446	23.121.73.156	443	203.106.151.150	52127	TCP	1	Not set	443 > 52127 [ACK] Seq=2897 Ack=1 Win=4121 Len=1448 TSval=3160388895
11	2017-11-19	05:00:00.460447	23.121.73.156	443	203.106.151.150	52127	TCP	1	Not set	443 > 52127 [ACK] Seq=4345 Ack=1 Win=4121 Len=1448 TSval=3160388895
12	2017-11-19	05:00:00.460448	23.121.73.156	443	203.106.151.150	52127	TCP	1	Not set	443 > 52127 [ACK] Seq=5793 Ack=1 Win=4121 Len=1448 TSval=3160388895
13	2017-11-19	05:00:00.460450	23.121.73.156	443	203.106.151.150	52127	TCP	1	Not set	443 > 52127 [ACK] Seq=7241 Ack=1 Win=4121 Len=1448 TSval=3160388895
14	2017-11-19	05:00:00.460452	173.194.175.99	443	163.222.158.37	54423	TCP	1	Not set	443 > 54423 [ACK] Seq=1 Ack=1 Win=228 Len=1432
15	2017-11-19	05:00:00.460453	173.194.175.99	443	163.222.158.37	54423	TCP	1	Not set	443 > 54423 [ACK] Seq=1433 Ack=1 Win=228 Len=1432
16	2017-11-19	05:00:00.460462	203.106.149.144	80	175.68.186.71	60754	TCP	1	Not set	80 > 60754 [PSH, ACK] Seq=1441 Ack=1 Win=64080 Len=1440 TSval=31115
17	2017-11-19	05:00:00.460468	216.165.42.17	443	163.222.245.147	58983	TCP	1	Not set	443 > 58983 [ACK] Seq=1 Ack=1 Win=193 Len=0
18	2017-11-19	05:00:00.460596	203.106.147.13	42269	45.55.114.223	81	ICMP	0	Not set	Time-to-live exceeded (Time to live exceeded in transit)
19	2017-11-19	05:00:00.460691	202.20.88.240	64193	13.11.252.112	80	TCP	1	Not set	64193 > 80 [ACK] Seq=1 Ack=1 Win=1681 Len=0 SLE=1461 SRE=61321
20	2017-11-19	05:00:00.460760	203.106.156.48	80	1.21.142.97	45640	TCP	1	Not set	80 > 45640 [ACK] Seq=1 Ack=1 Win=480 Len=1388 TSval=1721915590 TSec
21	2017-11-19	05:00:00.460783	203.106.156.48	80	1.21.142.97	45640	TCP	1	Not set	80 > 45640 [ACK] Seq=1389 Ack=1 Win=480 Len=1388 TSval=1721915590 T
22	2017-11-19	05:00:00.461118	139.141.130.92	40390	133.186.105.6	102	TCP	0	Not set	40390 > 102 [SYN] Seq=0 Win=65535 Len=0
23	2017-11-19	05:00:00.461228	173.194.175.99	443	163.222.158.37	54423	TCP	1	Not set	443 > 54423 [ACK] Seq=2865 Ack=1 Win=228 Len=1432
24	2017-11-19	05:00:00.461230	173.194.175.99	443	163.222.158.37	54423	TCP	1	Not set	443 > 54423 [ACK] Seq=4297 Ack=1 Win=228 Len=1432
25	2017-11-19	05:00:00.461439	58.222.40.29	7994	150.177.41.46	23	TCP	0	Not set	7994 > 23 [SYN] Seq=0 Win=8198 Len=0
26	2017-11-19	05:00:00.461491	89.8.159.99	49783	163.222.147.250	23	TCP	0	Not set	49783 > 23 [SYN] Seq=0 Win=14600 Len=0
27	2017-11-19	05:00:00.461705	89.7.83.236	20217	133.186.42.8	8333	TCP	0	Not set	20217 > 8333 [SYN] Seq=0 Win=9797 Len=0
28	2017-11-19	05:00:00.461727	203.106.149.144	80	222.72.19.218	39844	TCP	1	Not set	80 > 39844 [ACK] Seq=1 Ack=1 Win=64261 Len=1436 TSval=311154608 TSe
29	2017-11-19	05:00:00.461729	45.227.124.193	80	163.222.158.10	55610	TCP	1	Not set	80 > 55610 [ACK] Seq=2479 Ack=1 Win=1877 Len=1448 TSval=1292746861
30	2017-11-19	05:00:00.461731	45.227.124.193	80	163.222.158.10	55610	TCP	1	Not set	80 > 55610 [ACK] Seq=3927 Ack=1 Win=1877 Len=1448 TSval=1292746861

Figure 4.13: Successful upload of a packet trace file into database

4.3.2 Testing SYN Floods Attack Detection Algorithm

The main functional requirement for the developed application is to run or execute the proposed anomaly detection algorithm and its code. And here the proposed algorithm had a successful implementation and testing.

4.3.2.1 Evaluation of The SYN Floods Attacks Detected in Traffic Traces Captured on 15th Nov, 2017 through 24th Nov, 2017.

This section tests the SYN Floods detection algorithm on each of the network packet traces data in the database attacks table and then analyses the data for representation in a bar chart. Four-day network traffic packet trace was found SYN Floods attack out of ten-day capture. The detected SYN Floods attacks in each of the affected network packet traces captured from 15th – 24th November, 2017 are displayed on the Dashboard – output table as shown in Appendices I through M respectively.

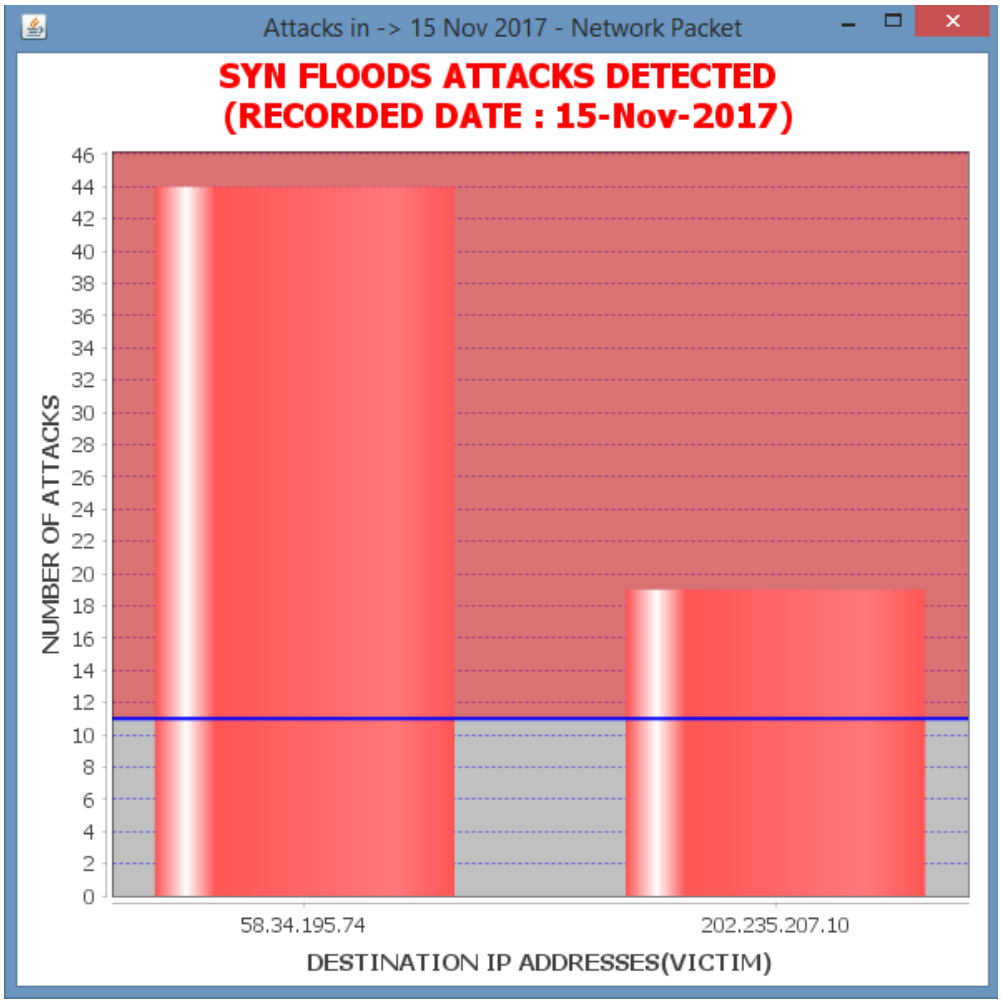


Figure 4.14: Bar chart representation of SYN Floods attacks detected in "*15 Nov, 2017 – Network Packet*" trace

In the chart of Figure 4.14, the following Destination IP addresses "58.34.195.74" and "202.235.207.10" are the victims that received "44" and "19" SYN floods attacks respectively. Also as shown in Appendix I, the distributed denials of service attack (DDoS) attackers sent multiple numbers of spoofed packets using different source IPs with the same source port number "22", targeting one particular host server (destination IP). These were mostly experienced in IP address "58.34.195.74". These attacks consequently vary range of host server ports which exhaust the resources of the target server and deny legitimate user the opportunity to connect. The attacks on "58.34.195.74" started at 05:00:00.270247 and ended at 05:00:01.111302 (as shown in Appendix I). It shows that the attacks occurred within the duration of 0.84 seconds. This shows that the attacker sends 1 request in every 0.2 second. Also attacks on "202.235.207.10" started at 05:00:00.296566 and ended at 05:00:01.176230

(see Appendix I), showing that attacks occurred within the duration of 0.88 seconds. It means that the attacker sends 1 request in every 0.05 seconds. This can also be seen in Appendix I.

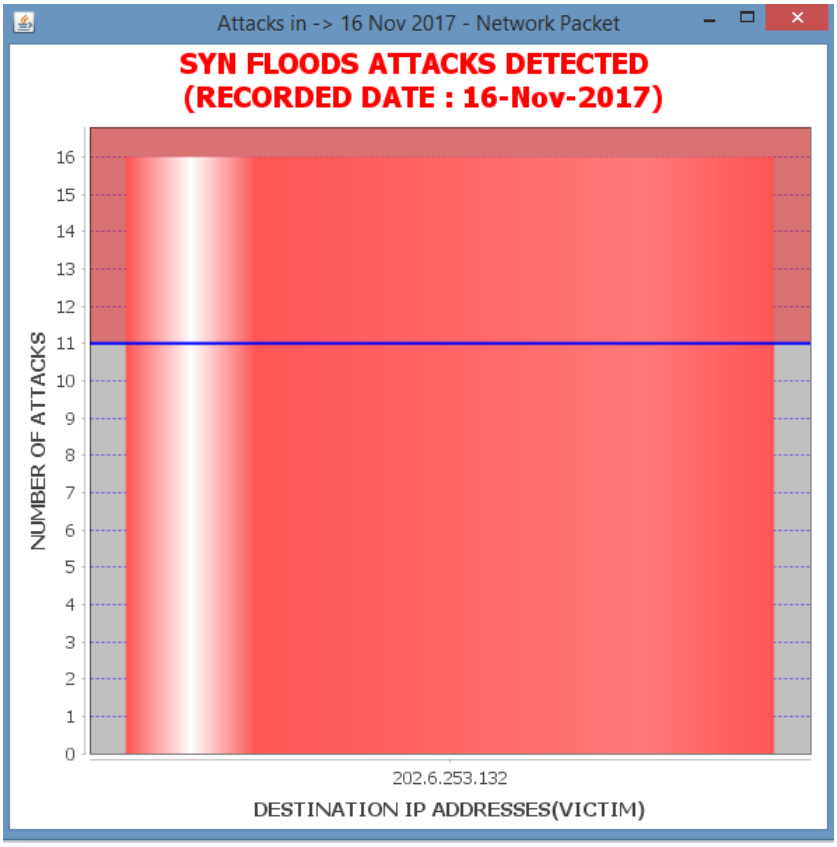


Figure 4.15: Bar chart representation of SYN Floods attacks detected in "*16 Nov 2017 – Network Packet*" trace

The bar chart of Figure 4.15 shows that the distributed denials of service (DDoS) attackers sent "16" spoofed packets using different source IP addresses with source ports number "443" and "80" (as shown in Appendix J), targeting one particular host server "202.6.253.132". Thereby varies host server ports with the aim to exhaust the resources of the server and deny legitimate user the opportunity to connect. The attacks on "202.6.253.132" started at 05:00:00.399328 and ended at 05:00:00.748758 (see Appendix J). It shows that the attacks occurred within the duration of 0.35 seconds. And also the attacker sends 1 packet request per 0.02 second. This can also be seen in Appendix J.

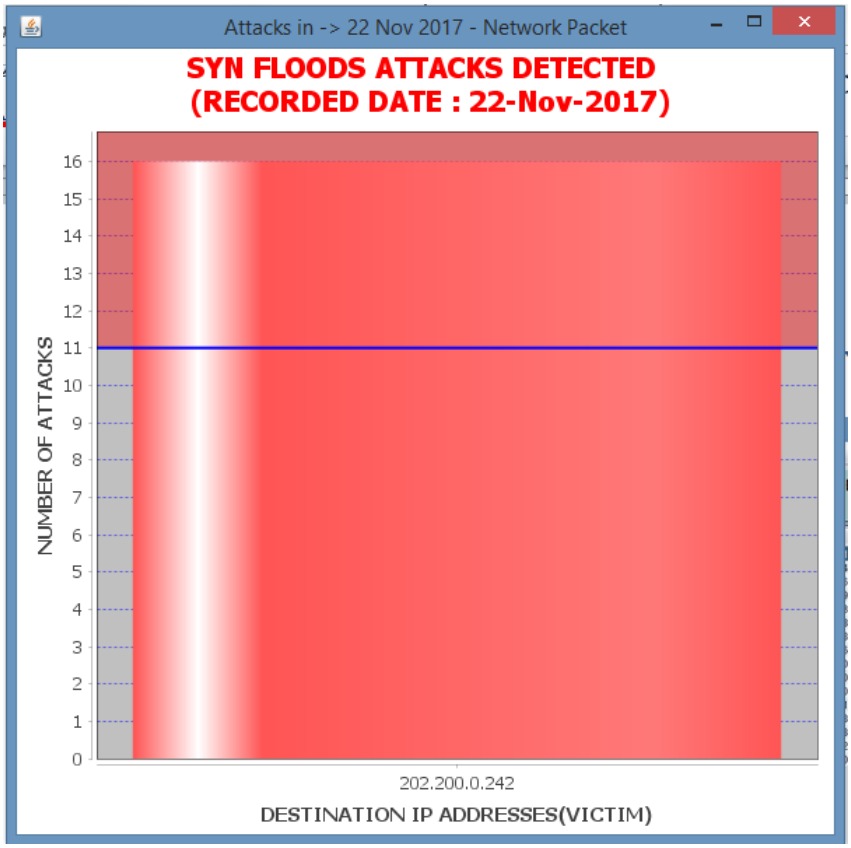


Figure 4.16: Bar chart representation of SYN Floods attacks detected in "*22 Nov 2017 – Network Packet*" trace

Appendix K shows that the distributed denials of service (DDoS) attackers sent "16" spoofed packets using different source IP addresses with the same source port number "443", targeting one particular host server "202.200.0.242". It thereby varies host server ports with the aim to exhaust the resources of the server and deny legitimate user the opportunity to connect. The attacks on "202.200.0.242" started at 05:00:00.415111 and ended at 05:00:00.904561. It implies that the attacks occurred within the duration of 0.49 seconds. And also the attacker sends 1 packet request per 0.03 second. This can also be seen in Appendix K.

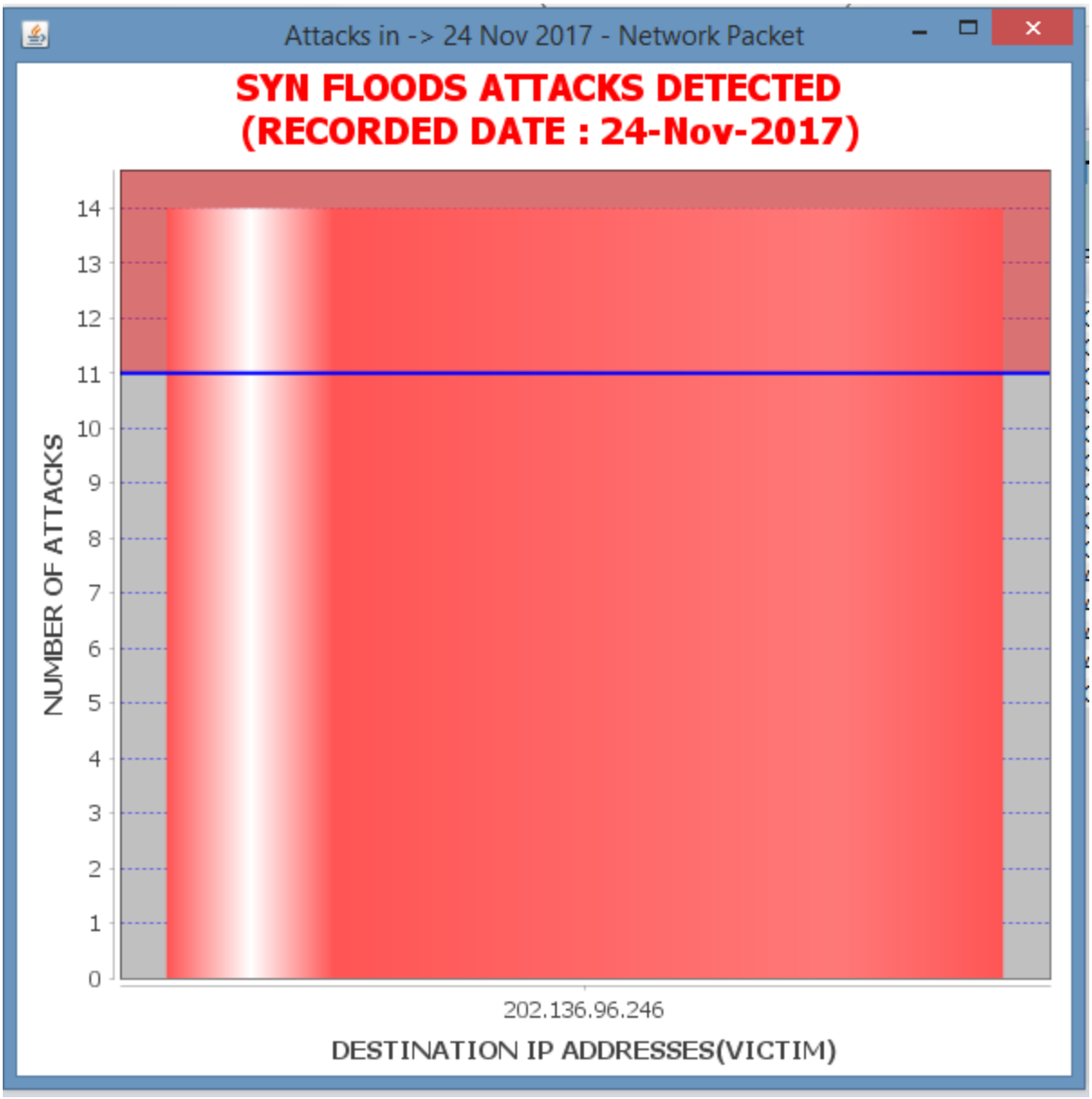


Figure 4.17: Bar chart representation of SYN floods attacks detected in "*24 Nov 2017 – Network Packet*" packet trace

The bar chart of Figure 4.17 shows the SYN Floods attacks detected in 24th Nov, 2017. The distributed denials of service (DDoS) attackers sent "14" spoofed packets using different source IP addresses with the same source port number "80" and "443" (as shown in Appendix M), targeting one particular host server "202.136.96.246" and thereby varying host server ports with the aim to exhaust the resources of the server and deny legitimate user the opportunity to connect. The attacks on "202.136.96.246" started at 05:00:00.590478 and ended at 05:00:00.769695 (see Appendix M). It implies that the attacks occurred within the

duration of 0.17 seconds. Also the attacker sends 1 packet request per 0.01 second. This can also be seen in Appendix M.

4.3.3 Summary of SYN Floods DDoS Attack Detection Algorithm

This section summarizes all the attacks detected in a simple graph for packet traces captured on 15th Nov 2017 through 24th Nov 2017. Figure 4.19 and Table 4.9 show how the proposed anomaly detection algorithm performed in detecting SYN Floods Distributed Denial of Service (DDoS) attack in any given network traffic packet.

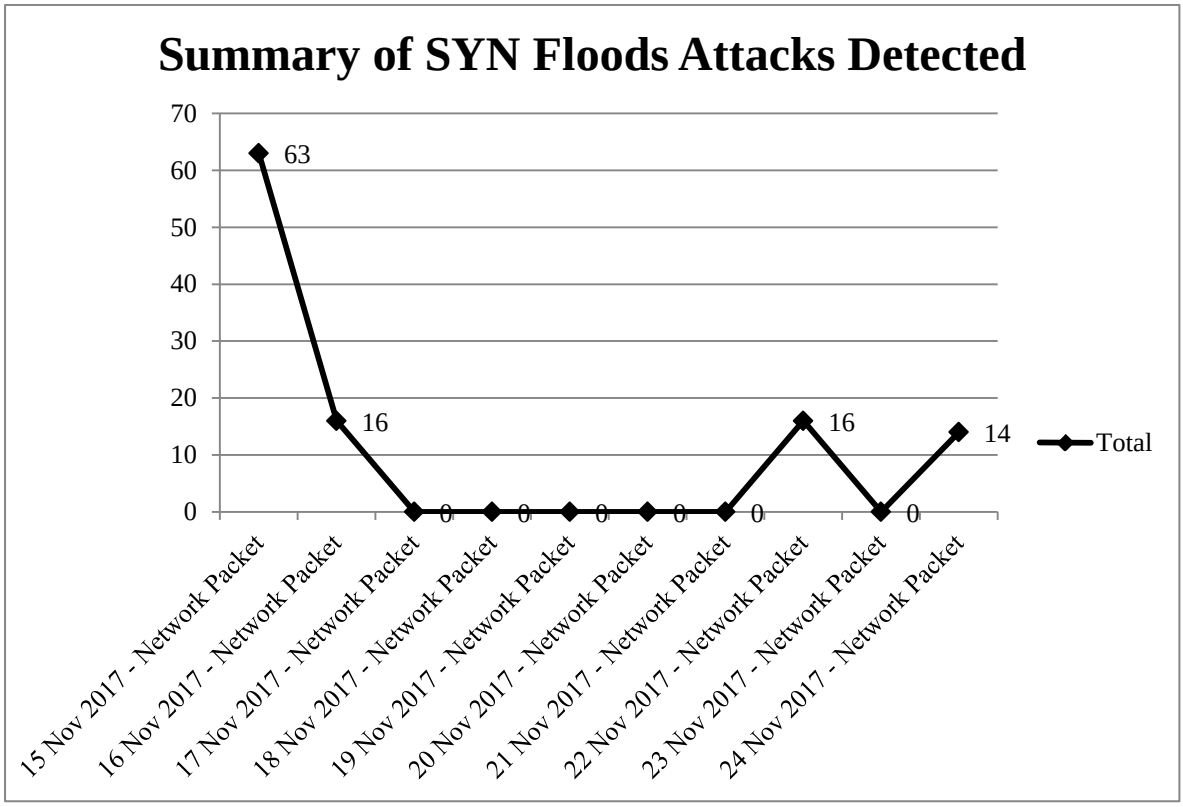


Figure 4.19: Summary of SYN Floods attacks detected from packet traces captured on 15th Nov 2017 through 24th Nov 2017

The graph represented in Figure 4.19 shows the packet trace datasets on the horizontal axis and total number of attacks in each packet traces datasets on the vertical axis. It displays the summary of SYS Floods DDoS attacks detected in each of the network packet traces. It can be observed that attacks were detected in only four datasets, which are network packet traces captured on 15th, 16th 22nd and 24th November 2017. While other six datasets were attack free. All the attacks detected occurred in less than 1 minutes. Table 4.9, presents the summary of the entire network traffic datasets used in this research work. It shows the packet trace names and number of network packets or conversation between client and host servers in each of the datasets. It also presents the

number of attacks and victims detected using the proposed SYN Floods detection algorithm (Java Logical Program).

Table 4.9: Detection summary table from packet traces captured from 15th through 24th November, 2017.

S/N	PACKET TRACE NAME	TCP Packets	No. of Attacks	Attack Duration	Frequency of an Attack Sent in Seconds	No. of Victims
1.	15 Nov 2017 – Network Packet	84,636	63	0.84, 0.88	0.2, 0.05	2
2.	16 Nov 2017 – Network Packet	45,977	16	0.35	0.02	1
3.	17 Nov 2017 – Network Packet	32,806	-	-	-	-
4.	18 Nov 2017 – Network Packet	22,440	-	-	-	-
5.	19 Nov 2017 – Network Packet	48,163	-	-	-	-
6.	20 Nov 2017 – Network Packet	52,900	-	-	-	-
7.	21 Nov 2017 – Network Packet	65,107	-	-	-	-
8.	22 Nov 2017 – Network Packet	48,423	16	0.49	0.03	1
9.	23 Nov 2017 – Network Packet	48,721	-	-	-	-
10.	24 Nov 2017 – Network Packet	44,949	14	0.17	0.01	1

4.4 VALIDATION OF THE PROPOSED ALGORITHM

The result obtained in this experiment using proposed JLP(Java Logical Program) SYN Floods DDoS attack detection algorithm, shows that SYN floods attack inundate a host server with [SYN] segment containing forged (spoofed) IP source addresses with non-existent or unreachable addresses. Host server responds with [SYN, ACK] segments to these addresses and then waits for responding acknowledgement [ACK] segments. Host server will not receive any acknowledgment [ACK] response and eventually time out. This is because the response was sent to non-existent or unreachable IP addresses. JLP algorithm detects those attacks flooding a host with incomplete TCP connection ([SYN] and [SYN, ACK] without an [ACK]). The detection algorithm result shows that the attacker eventually attempts filling the memory buffer of the victim. Once this buffer is full, the host can no longer process new TCP connection requests. This flood might even damage the victim's operating system. Either way, the attack disables the victim and its normal operations. Results show that the attacker sent request with multiple IP addresses of the same Port Number against a Victim IP address with varying Port Number or a single Port Number.

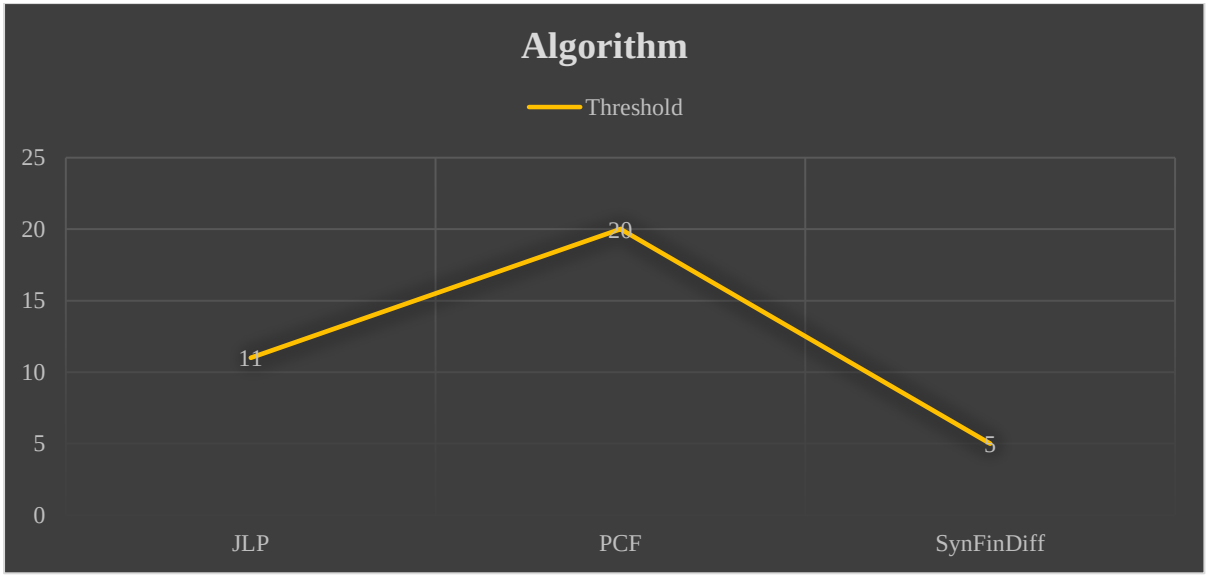


Figure 4.20: illustrates threshold evaluations of the existing and proposed algorithm

In figure 4.20, the threshold used in Java Logical Program (JLP), Partial Completion Filter (PCF) and SynFinDiff algorithms has a lot to contribute to the body of knowledge, many of the false alarms and minor anomalies seen by SynFinDiff and SynRate, spring from small increases in traffic during periods of light network activity.

This is because it uses a threshold traffic rate “5” below which any supposed attack is irrelevant. But for Java Logical program (JLP) algorithm setting this threshold at, say, 11 SYNs is a fairly conservative figure, and not reporting an attack for any period during which traffic is under the threshold would eliminate many of the false positives. In figure 4.20 Partial Completion Filter (PCF) sets its threshold to “20”, below which any supposed attack is irrelevant. But reporting an attack for any period during which traffic is under the threshold in PCF would eliminate many of the true positives.

Java Logical Program (JLP) algorithm reduces to the barest minimum the false positive alarm when TCP retransmission packets with half opened connection are detected. Such requests are ignored for a period during which its traffic is below threshold and should not be considered as an attack. The proposed anomaly detection algorithm detected attacks that have an anomalous behaviour from normal behaviour of TCP (Transport Control Protocols) characteristics which state that firstly, the client sends the first segment a, SYN segment to server, after receiving SYN segment from client; server sends a SYN+ACK segment back to client and then client responds with ACK segment and the connection is established.

CHAPTER FIVE

CONCLUSION

The primary objective of this research as earlier stated was to produce a functional algorithm for detecting anomalous traffic data in network Transport Control Protocols (TCP). This research work proposed 'Synchronize' Synchronization Packet Flood Distributed Denial of Service (SYN Flood DDoS) attacks detection algorithm on network Transport Control Protocols (TCP), in order to analyze and examine network traffic traces and see how this affect detecting anomalies in distributed attacks. This anomaly detection algorithm was developed using Object-Oriented Software Engineering (OOSE) methodology, which is compliant to Model-Driven Architecture (MDA) in the development processes.

5.1 SUMMARY

The adoption of Object-Oriented Software Engineering (OOSE) solution methodology of anomaly detection algorithm is a good approach for detecting SYN Floods DDoS attack in network traffic packet traces. This work started by analyzing existing research about data center archicture and traffic. In order to examine network traffic traces (datasets) and seeing how anomalies in Distributed Denial of Service attacks can be detected. Anomaly detection algorithm was proposed based on what traffic it analyzes and where in the cloud environment that traffic was collected. The data for anomaly detection comes from continuous monitoring at various levels in cloud such as (hypervisor and system level) and recording of measurements data. The algorithm deploy Entity-Relationship model in organizing the main information object. Java programing language was chosen for the development of the proposed algorithm. Finally, Detection of SYN Flood Distributed Denial of Service attacks were performed on network packet traces (datasets) captured from 15th through 24th November, 2017 to determine how well anomalies can be detected on TCP network protocols. The results obtained while testing the proposed algorithm summarizes SYS Flood DDoS attacks detected in each of the network packet traces.

5.2 KEY FINDINGS

The results obtained in this experiment using proposed JLP(Java Logical Program) SYN Floods DDoS attack detection algorithm, show that SYN floods attack inundate a host server with [SYN] segment containing forged (spoofed) IP source addresses with non-existent or

unreachable addresses. Host server responds with [SYN, ACK] segments to these addresses and then waits for responding acknowledgement [ACK] segments. Host server will not receive any acknowledgment [ACK] response and eventually time out. This is because the response was sent to non-existent or unreachable IP addresses. JLP algorithm detects those attacks flooding a host with incomplete TCP connection ([SYN] and [SYN, ACK] without an [ACK]). The detection algorithm result shows that the attacker eventually attempts filling the memory buffer of the victim. Once this buffer is full, the host can no longer process new TCP connection requests. This flood might even damage the victim's operating system. Either way, the attack disables the victim and its normal operations. Results show that the attacker sent request with multiple IP addresses of the same Port Number against a Victim IP address with varying Port Number or a single Port Number. It can be observed that attacks were detected in only four datasets, which are network packet traces captured on 15th, 16th 22nd and 24th November 2017. While other six datasets were attack free. All the attacks detected occurred in less than 1 minutes.

5.3 AUTHOR CONTRIBUTION TO THE BODY OF KNOWLEDGE

1. Produced a JLP (Java Logical Program) SYN Floods anomaly detection algorithm for network Transport Control Protocol (TCP) traffic data.
2. The established algorithm was made in a way, such that, it can be applied in any further research on TCP network protocols for SYN floods DDoS attacks detection.

5.4 FUTURE WORK

This research successfully provides and developed an algorithm for detecting SYN flood DDOS attack on network Transport Control Protocol (TCP). In future, we will like to analyze and provide solutions to other flooding attacks on network protocols such as UDP flooding, etc.

REFERENCE

[1] C.Lakshmi Devasena, Operations & Systems, ISB Hyderabad, IFHE University, Hyderabad. Impact study of cloud computing on business development, An International Journal (ORAJ), Vol. 1, No.1, August 2014

[2] Wireless 60 GHz Rack to Rack Communication in a Data Center Environment Francois, Avery John. Rochester Institute of Technology, ProQuest Dissertations Publishing, 2016. 10117667.

[3] Chandrasekaran, Siva Shankar, "Understanding Traffic Characteristics in a Server to Server Data Center Network" (2017). Thesis. Rochester Institute of Technology

[4] Tahiliani R., Tahiliani M., Sekaran K. (2015) TCP Congestion Control in Data Center Networks. In: Khan S., Zomaya A. (eds) Handbook on Data Centers. Springer, New York, NY.

[5] Kachris, Christoforos, and Ioannis Tomkos. "A survey on optical interconnects for data centers." *IEEE Communications Surveys & Tutorials* 14.4 (2012): 1021-1036.

[6] B. Vamanan, J. Hasan, and T. Vijaykumar. Deadline-aware datacenter TCP (D2TCP). SIGCOMM Comput. Commun. Rev., 42(4):115–126, Aug. 2012.

[7] Zhang X. (2016) A Flow Scheduling Algorithm Based on VM Migration in Data Center Networks. In: Huang X., Xiang Y., Li KC. (eds) Green, Pervasive, and Cloud Computing. Lecture Notes in Computer Science, vol 9663. Springer, Cham.

[8] Kai Chen, Ankit Singla, Atul Singh, Kishore Ramachandran, Lei Xu, Yueping Zhang and Xitao Wen. OSA: an optical switching architecture for data center networks with unprecedented flexibility. Published in:Journal IEEE/ACM Transactions on Networking (TON) archive Volume 22 Issue 2, April 2014 Pages 498-511 IEEE Press Piscataway, NJ, USA.

[9] Haitao Wu, Zhenqian Feng, and Chuanxiong Guo. ICTCP: Incast Congestion Control for TCP in Data-Center Networks,Published in: IEEE/ACM Transactions on Networking (Volume: 21, Issue: 2, April 2013)

[10] T. Benson, A. Akella, and D. A. Maltz. Network traffic characteristics of data centers in the wild. In Proceedings of the 10th ACM SIGCOMM conference on Internet measurement, IMC ’10, pages 267–280, New York, NY, USA, 2010. ACM.

[11] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron. Towards predictable datacentre networks. SIGCOMM Comput. Commun. Rev., 41(4):242–253, Aug. 2011.

[12] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley. Improving dat-acenter performance and robustness with multipath TCP. SIGCOMM Comput. Commun. Rev., 41(4):266–277, Aug. 2011.

[13] D. Brauckhoff, K. Salamatian, and M. May. A signal processing view on packet sampling and anomaly detection. In Proceedings of the 29th conference on Information communications, INFOCOM’10, pages 713–721, Piscataway, NJ, USA, 2010. IEEE Press.

[14] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken. The nature of data center traffic: measurements & analysis. In Proceedings of the 9th ACM

SIGCOMM conference on Internet measurement conference, IMC '09, pages 202–208, New York, NY, USA, 2009.ACM
[15] A. Li, L. Gu, and K. Xu. Fast anomaly detection in large data centers. Proc. of the 2010 IEEE Global Commuincations Conference, 2010.
[16] T. Benson, A. Anand, A. Akella, and M. Zhang. Understanding data center traffic characteristics. In Proceedings of the 1st ACM workshop on Research on enterprise networking, WREN '09, pages 65–72, New York, NY, USA, 2009. ACM.
[17] D. Brauckhoff, K. Salamatian, and M. May. A signal processing view on packet sampling and anomaly detection. In Proceedings of the 29th conference on Information communications, INFOCOM'10, pages 713–721, Piscataway, NJ, USA, 2010. IEEE Press.
[18] A. A. Mahimkar, H. H. Song, Z. Ge, A. Shaikh, J. Wang, J. Yates, Y. Zhang, and J. Emmons. Detecting the performance impact of upgrades in large operational networks. SIGCOMM Comput. Commun. Rev., 41(4):303–314, Aug. 2010.
[19] C.-W. Chang, A. Gerber, B. Lin, S. Sen, and O. Spatscheck. Network DVR: a programmable framework for application-aware trace collection. In Proceedings of the 11th international conference on Passive and active measurement, PAM'10, pages 191–200, Berlin, Heidel-berg, 2010. Springer-Verlag.
[20] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. ACM Comput. Surv., 41(3):15:1–15:58, July 2009.
[21] Sari, A. (2015) A Review of Anomaly Detection Systems in Cloud Networks and Survey of Cloud Security Measures in Cloud Storage Applications. Journal of Information Security, 6, 142-154. doi: 10.4236/jis.2015.62015.
[22] Marhas, M.K., Bhange, A. and Ajankar, P. (2012) Anomaly Detection in Network Traffic: A Statistical Approach. International Journal of IT, Engineering and Applied Sciences Research (IJIEASR), 1, 16-20.
[23] L. Zhang, S. Yu, D. Wu, and P. Watters, "A Survey on Latest Botnet Attack and Defense," Proc. of 10th Intl' Conference On Trust, Security and Privacy in Computing and Communications (TrustCom), IEEE, pp. 53-60, November 2011
[24] A. Mishra, B.B. Gupta, and R.C. Joshi, "A Comparative Study of Distributed Denial of Service Attacks, Intrusion Tolerance and Mitigation Techniques," Proc. of European Intelligence and Security Informatics Conference (EISIC), IEEE, pp. 286-289, September 2011.
[25] K. W. M. Ghazali, and R. Hassan, "Flooding Distributed Denial of Service Attacks-A Review," Journal of Computer Science 7 (8), Science Publications, 2011, pp. 1218-1223.
[26] H. Beitollahi, and G. Deconinck, "Denial of ServiceAttacks: A Tutorial,", Electrical Engineering Department (ESAT), University of Leuven, Technical Report: 08-2011-0115, August 2011.
[27] Kumar, S., Dutta, K. and Asati, A. (2015) Two Pass Port Scan Detection Technique Based on Connection Pattern and Status on Sampled Data. Journal of Computer and Communications, 3, 1-8
[28] [1]Adathakula Sree Deepthi, [2]Dr. K.Venkata Rao. Anomaly Detection Using Principal Component Analysis, IJCST Vol. 5, Issue 4, Oct - Dec 2014
[29] A. Patel, Q. Qassim, Z. Shukor, J. Nogueira, J. Júnior, C. Wills, and P. Federal, "Autonomic agent -based self-managed intrusion detection and prevention system," In Proceedings of the South African Information Security Multi-Conference pp. 223-234, 2011.

[30] J.-H. Lee, M.-W. Park, J.-H. Eom, and T.-M. Chung, “Multi-level intrusion detection system and log management in cloud computing,” Advanced Communication Technology (ICACT), 2011 13th International Conference pp. 552-555, 2011.
[31] K. Vieira, A. Schulter, C. Westphall, and C. Westphall, “Intrusion detection for grid and cloud computing,” IT Professional Magazine, vol. 12, no. 4, pp. 38, 2010.
[32] S. N. Dhage, and B. Meshram, “Intrusion detection system in cloud computing environment,” International Journal of Cloud Computing, vol. 1, no. 2-3, pp. 261-282, 2012.
[33] A. Bakshi, and Y. B. Dujodwala, “Securing Cloud from DDOS Attacks Using Intrusion Detection System in Virtual Machine,” Communication Software and Networks, 2010. ICCSN'10. Second International Conference on, pp. 260-264, 2010.
[34] C. Mazzariello, R. Bifulco, and R. Canonico, “Integrating a network IDS into an open source Cloud Computing environment,” 2010 Sixth International Conference on Information Assurance and Security, pp. 265 - 270, 2010.
[35] Gupta, P. Kumar, and A. Abraham, “A Profile Based Network Intrusion Detection and Prevention System for Securing Cloud Environment,” International Journal of Distributed Sensor Networks, vol. 2013, pp. 1-12, 2013.
[36] A. V. Dastjerdi, K. A. Bakar, and S. G. H. Tabatabaei, “Distributed Intrusion Detection in Clouds Using Mobile Agents,” 2009 Third International Conference on Advanced Engineering Computing and Applications in Sciences, pp. 175 - 180, 2009.
[37] P. Kannadiga, and M. Zulkernine, “DIDMA- A Distributed Intrusion Detection System Using Mobile Agents.pdf,” Sixth International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing and First ACIS International Workshop on Self-Assembling Wireless Network, pp. 238 - 245, 2005.
[38] C. Modi, D. Patel, B. Borisaniya, H. Patel, A. Patel, and M. Rajarajan, “A survey of intrusion detection techniques in Cloud,” Journal of Network and Computer Applications, vol. 36, no. 1, pp. 42-57, 2013.
[39] Y. Mehmood, M. A. Shibli, U. Habiba, and R. Masood, “Intrusion Detection System in Cloud Computing-challenges and opportunities,” 2013 2nd National Conference on Information Assurance (NCIA), pp. 59 - 66, 2013.
[40] S. Roschke, F. Cheng, and C. Meinel, “An Extensible and Virtualization-Compatible IDS Management Architecture,” Information Assurance and Security, 2009. IAS'09. Fifth International Conference on, vol. 2, pp. 130-134, 2009.
[41] A. S. Ibrahim, J. Hamlyn-Harris, J. Grundy, and M. Almorsy, “CloudSec- a security monitoring appliance for Virtual Machines in the IaaS cloud model,” 2011 5th International Conference on Network and System Security, pp. 113 - 120, 2011.
[42] M. P. K. Shelke, M. S. Sontakke, and A. D. Gawande., “Intrusion detection system for cloud computing,” International Journal of Scientific & Technology Research, pp. 67-71, 2012.
[43] C.-C. Lo, C.-C. Huang, and J. Ku, “A Cooperative Intrusion Detection System Framework for Cloud Computing Networks,” 2010 39th International Conference on Parallel Processing Workshops, pp. 280-284, 2010.

[44] J. He, C. Tang, Y. Yang, Y. Qiao, and C. Liu, “3D-IDS: IaaS User-oriented Intrusion Detection System,” Information Science and Engineering (ISISE), 2012 International Symposium on, pp. 12-15, 2012.
[45] X. Zhao, B. Kevin, and P. Atul, “Virtual Machine Security Systems,” Advances in Computer Science and Engineering, pp. 339-365, 2009.
[46] D. Sher. "Gartner: Application Layer DDoS Attacks to Increase in 2013," https://www.corero.com/blog/361gartner-application-layer-ddos-attacks-to-increase-in-2013.html.
[47] "Riverhead WP "; http://www.cse.msu.edu/~cse825/Riverhead_WP.pdf.
[48] Haining Wang, Danlu Zhang, and Kang G. Shin. Detecting SYN flooding attacks. In Proc. of INFOCOM. IEEE Communications Society, 2002.
[49] B. E. Brodsky and B. S. Darkhovsky. Nonparametric Methods in Change-Point Problems. Kluwer Academic Publishers, 1993.
[50] Vasilios A. Siris and Fotini Papagalou. Application of anomaly detection algorithms for detecting SYN flooding attacks. In Proc. of Globecom. IEEE Communications Society, 2004.
[51] Ramana Rao Kompella, Sumeet Singh, and George Varghese. On scalable attack detection in the network. In Proc. of Internet Measurement Conference. ACM SIGCOMM, 2004.
[52] VINTON G. CERF AND ROBERT E. KAHN. A Protocol for Packet Network Intercommunication, 1974 IEEE. Reprinted, with permission, from IEEE Trans on Comms, Vol Com-22, and No 5 May 1974
[53] Stefano Ceri, Piero Fraternali, Aldo Bongio, Marco Brambilla, Sara Comai, Maristella Matera- Designing Data-Intensive Web Applications. 2003. ISBN: 1–55860–843–5, Publisher: Elsevier Science [USA].
[54] "Most Widely Deployed SQL Database Estimates". SQLite.org. Retrieved May 11, 2011.
[55] Gustavo Rossi. Oscar Pastor. Daniel Schwabe. Luis Olsina [Eds]: Web Engineering: Modelling and Implementation. 2008. ISBN: 978-1-84628-922-4, Publisher: SpringerVerlag London Limited 2008.
[56] MAWI Working Group Traffic Archive. http://mawi.wide.ad.jp/mawi/. Accessed Nov 2017.

APPENDIX A: FUNCTIONALITY CODE FOR "FETCH DATA FROM DATABASE" SUBMENU

Fetch Data from Database sub-menu - Functionality Implementation

```
private void
fetchDataFromDatabase_MenuActionPerformed(java.awt.event.ActionEvent evt) {
 Statement myQuery = null;
   ResultSet rsQuery = null;
   Connection conn = dbconnection.ConnectDB();
   model.setRowCount(0);
    String fileName = String.valueOf(packetName.getSelectedItem());
   try{
         notification.setText("Loading...");

if(packetName.getSelectedIndex()==0){JOptionPane.showMessageDialog(null,"Please
select packet trace name","Error Alert !!!", JOptionPane.ERROR_MESSAGE);}
       else{
        myQuery = conn.createStatement();
       rsQuery = myQuery.executeQuery("select * from packet_traces where
file_name='"+fileName+"'");
       int l = 0;
        while(rsQuery.next()){
           Long IPs = rsQuery.getLong("source_ip");
           Long IPd = rsQuery.getLong("destination_ip");
        model.insertRow(model.getRowCount(), new Object[]{
                    ++l,
                    rsQuery.getInt("no"),
                    rsQuery.getString("time"),
                    ((IPs >> 24 ) & 0xFF) + "." + ((IPs >> 16 ) & 0xFF) + "." + ((IPs >> 8 ) &
0xFF) + "." +( IPs & 0xFF),
                    rsQuery.getInt("source_port"),
                    ((IPd >> 24 ) & 0xFF) + "." + ((IPd >> 16 ) & 0xFF) + "." + ((IPd >> 8 )
& 0xFF) + "." +( IPd & 0xFF),
                    rsQuery.getInt("destination_port"),
                    rsQuery.getString("protocol"),
                    rsQuery.getString("ack"),
                    rsQuery.getString("fin"),
                    rsQuery.getString("info")
                  });
        }
       JOptionPane.showMessageDialog(null,"Task completed !!!");
       notification.setText(" ");
       DashboardSubmenu.setSelectedIndex(1);
       }
```

```
   }catch (SQLException se) //catch block
   {
      //Handle errors for JDBC
      se.printStackTrace();
   } catch (Exception e) //catch block
   {
      //Handle errors for Class.forName
      e.printStackTrace();
   } finally //finally block
   {
      //finally block used to close resources
      try //try block
      {
         if (conn != null)//condition
         {
            conn.close(); //close connection
         }
      } catch (SQLException se)//Handle errors
      {
         se.printStackTrace();
      }//end finally try
   }//end try
   System.out.println("Goodbye!"); //print on console



}
```

APPENDIX B: FUNCTIONALITY CODE FOR "EXPORT ATTACKS RESULT TO CSV FILE" SUBMENU

Export attacks result to CSV file

```
public class WriteToFileCsv
 {
  public void convertToCsv() throws IOException
  {
   Statement myQuery = null;
   ResultSet rsQuery = null;
   Connection conn = dbconnection.ConnectDB();
   String fileName = String.valueOf(packetName.getSelectedItem());
   FileWriter fw = new FileWriter("Result-"+fileName+".csv");
   PrintWriter out = new PrintWriter(fw);
    try{

       notification.setText("Exporting Result to CSV File...");

if(packetName.getSelectedIndex()==0){JOptionPane.showMessageDialog(null,"Please
select packet trace name","Error Alert !!!", JOptionPane.ERROR_MESSAGE);}
       else{
           out.print("S/N");
           out.print(",");
           out.print("No.");
           out.print(",");
           out.print("Time");
           out.print(",");
           out.print("Source");
           out.print(",");
           out.print("Source Port");
           out.print(",");
```

```
            out.print("Destination");
            out.print(",");
            out.print("Destination Port");
            out.print(",");
            out.print("Protocol");
            out.print(",");
             out.print("Length");
            out.print(",");
            out.print("Arrival Time");
            out.print(",");
            out.print("Acknowledgment");
            out.print(",");
            out.print("Fin");
            out.print(",");
            out.println("Info");
         myQuery = conn.createStatement();
        rsQuery = myQuery.executeQuery("select * from attacks where
file_name='"+fileName+"'");
        int l = 0;
         while(rsQuery.next()){
            Long IPs = rsQuery.getLong("source_ip");
            Long IPd = rsQuery.getLong("destination_ip");
    // ',' divides the word into columns


                    out.print(++l);
                    out.print(",");
                    out.print(rsQuery.getInt("no"));
                    out.print(",");
                    out.print(rsQuery.getString("time"));
                    out.print(",");
                    out.print(((IPs >> 24 ) & 0xFF) + "." + ((IPs >> 16 ) & 0xFF) + "." + ((IPs
>> 8 ) & 0xFF) + "." +( IPs & 0xFF));
```

```
                out.print(",");
                out.print(rsQuery.getInt("source_port"));
                out.print(",");
                out.print(((IPd >> 24 ) & 0xFF) + "." + ((IPd >> 16 ) & 0xFF) + "." +
((IPd >> 8 ) & 0xFF) + "." +( IPd & 0xFF));
                out.print(",");
                out.print(rsQuery.getInt("destination_port"));
                out.print(",");
                out.print(rsQuery.getString("protocol"));
                out.print(",");
                out.print(rsQuery.getString("length"));
                out.print(",");
                out.print(rsQuery.getString("arrival_time").replace(", ", " "));
                out.print(",");
                out.print(rsQuery.getString("ack"));
                out.print(",");
                out.print(rsQuery.getString("fin"));
                out.print(",");
                out.println(rsQuery.getString("info"));



        }
  //Flush the output to the file
  out.flush();

  //Close the Print Writer
  out.close();

  //Close the File Writer
  fw.close();
        JOptionPane.showMessageDialog(null,"Task completed !!!");
        notification.setText(" ");
```

```
        }


    }catch (SQLException se) //catch block
    {
        //Handle errors for JDBC
        se.printStackTrace();
    } catch (Exception e) //catch block
    {
        //Handle errors for Class.forName
        e.printStackTrace();
    } finally //finally block
    {
        //finally block used to close resources
        try //try block
        {
            if (conn != null)//condition
            {
                conn.close(); //close connection
            }
        } catch (SQLException se)//Handle errors
        {
            se.printStackTrace();
        }//end finally try
    }//end try
    System.out.println("Goodbye!"); //print on console


}
private void exportResultToCSVActionPerformed(java.awt.event.ActionEvent evt) {
WriteToFileCsv callCsv = new WriteToFileCsv(); try { callCsv.convertToCsv();
        } catch (IOException ex) {
Logger.getLogger(Dashboard.class.getName()).log(Level.SEVERE, null, ex);  }}
```

APPENDIX C: FUNCTIONALITY CODE FOR “DELETE PCKET TRACE FROM DATABASE” SUBMENU

Delete packet trace from database

```
private void deletePacketTraceActionPerformed(java.awt.event.ActionEvent evt) {
Connection conn = dbconnection.ConnectDB();
   //Connection conn = mysqlConnector.ConnectDB();
   String fileName = String.valueOf(packetName.getSelectedItem());
   String Date =
((JTextField)recordedDate_control.getDateEditor().getUiComponent()).getText();
    SimpleDateFormat dFormat = new SimpleDateFormat("dd-MMM-yyyy");
    String Date2 = dFormat.format(recordedDate_control.getDate());

   try{
         notification.setText("Deleting...");
      if((packetName.getSelectedIndex()==0 && Date.isEmpty()==true) ||
(packetName.getSelectedIndex()==0 || Date.isEmpty()==true)){
         notification.setText("Task Failed !!! ");
         if(packetName.getSelectedIndex()>0 && Date.isEmpty()==true)
         {
         JOptionPane.showMessageDialog(null,"Please Choose the Trace Record Date
!!","Error Alert !!!", JOptionPane.ERROR_MESSAGE);
         }

         if(packetName.getSelectedIndex()==0 && Date.isEmpty()==true)
         {
         JOptionPane.showMessageDialog(null,"Please select packet trace name and
Trace Record Date !!","Error Alert !!!", JOptionPane.ERROR_MESSAGE);
         }
         if(packetName.getSelectedIndex()==0 && Date.isEmpty()==false)
         {
         JOptionPane.showMessageDialog(null,"Please select packet trace name
```

```
!!","Error Alert !!!", JOptionPane.ERROR_MESSAGE);
          }
          }
       else{
        String sqlx = "DELETE FROM packet_traces where file_name='"+fileName+"'
and recordedDate='"+Date2+"'";
                    PreparedStatement pstmtx = conn.prepareStatement(sqlx);
                    // execute the delete statement
                    pstmtx.executeUpdate();
                    JOptionPane.showMessageDialog(null,fileName+" Deleted Successfully
!!!");
                    notification.setText(" ");
       }
    }catch (SQLException se) //catch block
    {
       //Handle errors for JDBC
      se.printStackTrace();
    } catch (Exception e) //catch block
    {
       //Handle errors for Class.forName
       e.printStackTrace();
    } finally //finally block
    {
       //finally block used to close resources
       try //try block
       {
          if (conn != null)//condition
          { conn.close(); //close connection }
       } catch (SQLException se)//Handle errors
       {  se.printStackTrace();}//end finally try
    }//end try
    System.out.println("Goodbye!"); //print on console }
```

APPENDIX D: FUNCTIONALITY CODE FOR "DELETE ATTACK HISTORY" SUBMENU

Delete attack history

```
private void deleteDetectedAttackHistoryActionPerformed(java.awt.event.ActionEvent
evt) {
Connection conn = dbconnection.ConnectDB();

   try{
      String sqlx = "DELETE FROM attacks";
                PreparedStatement pstmtx = conn.prepareStatement(sqlx);
                // execute the delete statement
                pstmtx.executeUpdate();
                JOptionPane.showMessageDialog(null,"All Attack History Deleted
Successfully !!!");
   }catch (SQLException se) //catch block
   {
      //Handle errors for JDBC
     se.printStackTrace();
   } catch (Exception e) //catch block
   {
      //Handle errors for Class.forName
      e.printStackTrace();
   } finally //finally block
   {
      //finally block used to close resources
      try //try block
      {
         if (conn != null)//condition
         {
            conn.close(); //close connection
         }
```

```
        } catch (SQLException se)//Handle errors
        {
            se.printStackTrace();
        }//end finally try
    }//end try
    System.out.println("Goodbye!"); //print on console


}
```

APPENDIX E: FUNCTIONALITY CODE FOR "DETECT SYN FLOODS ATTACKS" SUBMENU

SYN Floods DDoS Attack detection

```
private void SYN_FloodsAttackDetectionActionPerformed(java.awt.event.ActionEvent
evt) {
   Statement myQuery = null;
   Statement myQueryCh = null;
   Statement myQueryx = null;
   ResultSet rsQuery = null;
   ResultSet rsQueryx = null;
   ResultSet rsQueryCh = null;
   Connection conn = dbconnection.ConnectDB();
   //Connection conn = mysqlConnector.ConnectDB();
   model.setRowCount(0);
    String fileName = String.valueOf(packetName.getSelectedItem());
    String Date =
((JTextField)recordedDate_control.getDateEditor().getUiComponent()).getText();
       SimpleDateFormat dFormat = new SimpleDateFormat("dd-MMM-yyyy");
       String Date2 = dFormat.format(recordedDate_control.getDate());

    int ids=0;

   try{
      if((packetName.getSelectedIndex()==0 && Date.isEmpty()==true) ||
(packetName.getSelectedIndex()==0 || Date.isEmpty()==true)){
          notification.setText("Task Failed !!! ");
          if(packetName.getSelectedIndex()>0 && Date.isEmpty()==true)
          {
          JOptionPane.showMessageDialog(null,"Please Choose the Trace Record Date
!!","Error Alert !!!", JOptionPane.ERROR_MESSAGE);
          }
```

```
        if(packetName.getSelectedIndex()==0 && Date.isEmpty()==true)
        {
        JOptionPane.showMessageDialog(null,"Please select packet trace name and
Trace Record Date !!","Error Alert !!!", JOptionPane.ERROR_MESSAGE);
        }
        if(packetName.getSelectedIndex()==0 && Date.isEmpty()==false)
        {
        JOptionPane.showMessageDialog(null,"Please select packet trace name
!!","Error Alert !!!", JOptionPane.ERROR_MESSAGE);
        }
        }
      else{
         notification.setText("Detecting SYN Floods attacks...");
       myQuery = conn.createStatement();
       myQueryx = conn.createStatement();
       String sqlx = "DELETE FROM attacks where file_name='"+fileName+"' and
recordedDate='"+Date2+"'";
                  PreparedStatement pstmtx = conn.prepareStatement(sqlx);
                  // execute the delete statement
                  pstmtx.executeUpdate();
      rsQuery = myQuery.executeQuery("select * from packet_traces where
file_name='"+fileName+"' and info like '%[SYN, ACK]%' and
recordedDate='"+Date2+"'");
       while(rsQuery.next()){
         Long IPs = rsQuery.getLong("source_ip");
          Long IPd = rsQuery.getLong("destination_ip");

      myQueryCh = conn.createStatement();
      rsQueryCh = myQueryCh.executeQuery("select * from attacks order by id desc
limit 0,2");
       if(rsQueryCh.next()){
```

```
        // JOptionPane.showMessageDialog(null,"Last ID: "+ rsQuery.getInt("id"));
          ids = rsQuery.getInt("id")+1;

   String sql = "insert into
attacks(id,no,time,source_ip,source_port,destination_ip,destination_port,protocol,lengt
h,arrival_time,ack,fin,info,file_name,recordedDate) values(?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)";
   PreparedStatement statement = conn.prepareStatement(sql);
      statement.setInt(1, ids);
      statement.setInt(2, rsQuery.getInt("no"));
      statement.setString(3, rsQuery.getString("time"));
      statement.setLong(4,IPs);
      statement.setInt(5, rsQuery.getInt("source_port"));
      statement.setLong(6, IPd);
      statement.setInt(7, rsQuery.getInt("destination_port"));
      statement.setString(8, rsQuery.getString("protocol"));
      statement.setInt(9, rsQuery.getInt("length"));
      statement.setString(10, rsQuery.getString("arrival_time"));
      statement.setString(11, rsQuery.getString("ack"));
      statement.setString(12,  rsQuery.getString("fin"));
      statement.setString(13, rsQuery.getString("info"));
      statement.setString(14, rsQuery.getString("file_name"));
      statement.setString(15, rsQuery.getString("recordedDate"));
      statement.executeUpdate();
       }else{
      String sql = "insert into
attacks(id,no,time,source_ip,source_port,destination_ip,destination_port,protocol,lengt
h,arrival_time,ack,fin,info,file_name,recordedDate) values(?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)";
      PreparedStatement statement = conn.prepareStatement(sql);
      statement.setInt(1, ids);
      statement.setInt(2, rsQuery.getInt("no"));
      statement.setString(3, rsQuery.getString("time"));
      statement.setLong(4,IPs);
```

```
        statement.setInt(5, rsQuery.getInt("source_port"));
        statement.setLong(6, IPd);
        statement.setInt(7, rsQuery.getInt("destination_port"));
        statement.setString(8, rsQuery.getString("protocol"));
        statement.setInt(9, rsQuery.getInt("length"));
        statement.setString(10, rsQuery.getString("arrival_time"));
        statement.setString(11, rsQuery.getString("ack"));
        statement.setString(12,  rsQuery.getString("fin"));
        statement.setString(13, rsQuery.getString("info"));
        statement.setString(14, rsQuery.getString("file_name"));
        statement.setString(15, rsQuery.getString("recordedDate"));
        statement.executeUpdate();
        }

        }

        rsQueryx = myQueryx.executeQuery("select * from packet_traces where
file_name='"+fileName+"' and info like '%[ACK]%' and
recordedDate='"+Date2+"'");
            while(rsQueryx.next()){
               Long IPs_2 = rsQueryx.getLong("source_ip");
                Long IPd_2 = rsQueryx.getLong("destination_ip");



                    String sql = "DELETE FROM attacks WHERE file_name=? and
source_ip=? and source_port=? and destination_ip=? and destination_port=? and
recordedDate=? or source_ip=? and source_port=? and destination_ip=? and
destination_port=? and recordedDate=?";

                    PreparedStatement pstmt = conn.prepareStatement(sql);
                    // set the corresponding param
                    pstmt.setString(1, fileName);
```

```
                pstmt.setLong(2, IPs_2);
                pstmt.setInt(3, rsQueryx.getInt("source_port"));
                pstmt.setLong(4, IPd_2);
                pstmt.setLong(5, rsQueryx.getLong("destination_port"));
                pstmt.setString(6, rsQueryx.getString("recordedDate"));
                pstmt.setLong(7, IPd_2);
                pstmt.setInt(8, rsQueryx.getInt("destination_port"));
                pstmt.setLong(9, IPs_2);
                pstmt.setInt(10, rsQueryx.getInt("source_port"));
                pstmt.setString(11, rsQueryx.getString("recordedDate"));
              // execute the delete statement
                pstmt.executeUpdate();
        }
        rsQueryx = myQueryx.executeQuery("select * from attacks where
file_name='"+fileName+"' and recordedDate='"+Date2+"'");
        int k = 0;
        while(rsQueryx.next()){
           Long IPs_2 = rsQueryx.getLong("source_ip");
            Long IPd_2 = rsQueryx.getLong("destination_ip");
           model.insertRow(model.getRowCount(), new Object[]{
                        ++k,
                        rsQueryx.getInt("no"),
                        rsQueryx.getString("time"),
                        (( IPs_2 >> 24 ) & 0xFF) + "." + (( IPs_2 >> 16 ) & 0xFF) + "." +((
IPs_2 >> 8 ) & 0xFF) + "." +(  IPs_2 & 0xFF),
                        rsQueryx.getInt("source_port"),
                        ((IPd_2 >> 24 ) & 0xFF) + "." + ((IPd_2 >> 16 ) & 0xFF) + "." +
((IPd_2 >> 8 ) & 0xFF) + "." +( IPd_2 & 0xFF),
                        rsQueryx.getInt("destination_port"),
                        rsQueryx.getString("protocol"),
                        rsQueryx.getString("ack"),
                        rsQueryx.getString("fin"),
```

```
                            rsQueryx.getString("info")
                        });
            }
        JOptionPane.showMessageDialog(null, "Task completed!!!");
        notification.setText(" ");
         DashboardSubmenu.setSelectedIndex(1);
       }
    }catch (SQLException se) //catch block
    {
       //Handle errors for JDBC
       se.printStackTrace();
    } catch (Exception e) //catch block
    {
       //Handle errors for Class.forName
       e.printStackTrace();
    } finally //finally block
    {
       //finally block used to close resources
       try //try block
       {
          if (conn != null)//condition
          {
             conn.close(); //close connection
          }
       } catch (SQLException se)//Handle errors
       {
          se.printStackTrace();
       }//end finally try
    }//end try
    System.out.println("Goodbye!"); //print on console
}
```

APPENDIX F: FUNCTIONALITY CODE FOR "DETECTED ATTACK HISTORY" SUBMENU

Detected Attack History

```
private void detectedAttackHistoryActionPerformed(java.awt.event.ActionEvent evt) {
   Connection conn = dbconnection.ConnectDB();
   Statement myQuery = null;
   Statement myQueryx = null;
   ResultSet rsQuery = null;
   ResultSet rsQueryx = null;
   String fileName = String.valueOf(packetName.getSelectedItem());
   String Date =
((JTextField)recordedDate_control.getDateEditor().getUiComponent()).getText();
  SimpleDateFormat dFormat = new SimpleDateFormat("dd-MMM-yyyy");
    String Date2 = dFormat.format(recordedDate_control.getDate());

 try{
    if((packetName.getSelectedIndex()==0 && Date.isEmpty()==true) ||
(packetName.getSelectedIndex()==0 || Date.isEmpty()==true)){
          notification.setText("Task Failed !!! ");
          if(packetName.getSelectedIndex()>0 && Date.isEmpty()==true)
          {
          JOptionPane.showMessageDialog(null,"Please Choose the Trace Record Date
!!","Error Alert !!!", JOptionPane.ERROR_MESSAGE);
          }

          if(packetName.getSelectedIndex()==0 && Date.isEmpty()==true)
          {
          JOptionPane.showMessageDialog(null,"Please select packet trace name and
Trace Record Date !!","Error Alert !!!", JOptionPane.ERROR_MESSAGE);
          }
          if(packetName.getSelectedIndex()==0 && Date.isEmpty()==false)
```

```
        {
        JOptionPane.showMessageDialog(null,"Please select packet trace name
!!","Error Alert !!!", JOptionPane.ERROR_MESSAGE);
        }
        }
      else{
      DefaultCategoryDataset detectionChart = new DefaultCategoryDataset();
      myQuery = conn.createStatement();
      myQueryx = conn.createStatement();
      rsQuery = myQuery.executeQuery("select * from attacks where
file_name='"+fileName+"' and recordedDate='"+Date2+"' group by destination_ip");
      ArrayList<String> numVictimCount = new ArrayList<String>();

      while(rsQuery.next())
        {
         /* if(rsQuery.last()){
             System.out.print(rsQuery.getLong("destination_ip")+" =>
"+rsQuery.getRow());

          } */
             Long IPd = rsQuery.getLong("destination_ip");
              String dest = ((IPd >> 24 ) & 0xFF) + "." + ((IPd >> 16 ) & 0xFF) + "." +
((IPd >> 8 ) & 0xFF) + "." +( IPd & 0xFF);

            rsQueryx = myQueryx.executeQuery("select * from attacks where
file_name='"+fileName+"' and destination_ip='"+IPd+"' and
recordedDate='"+Date2+"'");
            ArrayList<String> list= new ArrayList<String>();
             while(rsQueryx.next())
             {
              list.add(rsQueryx.getString("source_ip"));
             }
```

```
            if(list.size()<=10){
                //detectionChart.setValue(list.size(), "Marks", "False Positive");
             }else{
                detectionChart.setValue(list.size(), "Marks", dest);
                numVictimCount.add(dest);
             }

             System.out.println(list.size()+" => "+dest);

          }
  JFreeChart chart = ChartFactory.createBarChart("SYN FLOODS ATTACKS
DETECTED \n (RECORDED DATE : "+Date2+")", "DESTINATION IP
ADDRESSES(VICTIM)", "NUMBER OF ATTACKS", detectionChart,
PlotOrientation.VERTICAL, false, false, false);
          chart.setBackgroundPaint(Color.LIGHT_GRAY);
          chart.getTitle().setPaint(Color.red);
          chart.setBackgroundPaint(Color.WHITE);
          chart.setBorderPaint(Color.ORANGE);
          CategoryPlot p = chart.getCategoryPlot();
          p.setRangeGridlinePaint(Color.BLUE);
                ValueMarker vm = new ValueMarker(11, Color.BLUE, new
BasicStroke(2.0f));
                p.addRangeMarker(vm);
                 IntervalMarker target = new IntervalMarker(11, 10000, new Color(1, 0,
0, 1/2f));
                 p.addRangeMarker(target, Layer.BACKGROUND);

          /*  String[] columnNames =
          {"Destination IP","Number of Attack(s)","Duration"};
          Object[][] data =
          {
            {"32.115.242.10", "465", "10 sec"},
```

```
            {"365.119.242.10", "464", "20 sec"},
            {"102.115.242.10", "475", "1 sec"},
            {"32.115.242.10", "465", "4 sec"}
        };
        JTable outputTable = new JTable(data, columnNames);
        outputTable.setVisible(true);
       JScrollPane tableScrollPane = new JScrollPane(outputTable);
        tableScrollPane.setVisible(true);
        tableScrollPane.setSize(1200, 600);
        tableScrollPane.setBounds(200, 20, 1200, 600); */

        ChartFrame frame = new ChartFrame("Attacks in -> "+fileName,chart);
        frame.setVisible(true);
        if(numVictimCount.size()<=3){
          frame.setSize(600, 600);
          frame.setBounds(400, 20, 600, 600);
        }else{
          frame.setSize(1000, 600);
          frame.setBounds(200, 20, 1000, 600);
        }
        frame.setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE);


  }
}catch (SQLException se) //catch block
 {
    //Handle errors for JDBC
   se.printStackTrace();
 } catch (Exception e) //catch block
 {
    //Handle errors for Class.forName
    e.printStackTrace();
```

```
} finally //finally block
{
   //finally block used to close resources
   try //try block
   {
      if (conn != null)//condition
      {
         conn.close(); //close connection
      }
   } catch (SQLException se)//Handle errors
   {
      se.printStackTrace();
   }//end finally try
}//end try
System.out.println("Goodbye!"); //print on console


}
```

APPENDIX G: FUNCTIONALITY CODE FOR "CHART REPRESENTATION OF SYN FLOODS ATTACKS DETECTED IN BAR CHART" SUBMENU

Chart Representation of SYN Floods attacks detection in bar chart

```
private void
attackRepresentationInBarChartActionPerformed(java.awt.event.ActionEvent evt) {
   Connection conn = dbconnection.ConnectDB();
   Statement myQuery = null;
   Statement myQueryx = null;
   ResultSet rsQuery = null;
   ResultSet rsQueryx = null;
   String fileName = String.valueOf(packetName.getSelectedItem());
   String Date =
((JTextField)recordedDate_control.getDateEditor().getUiComponent()).getText();
  SimpleDateFormat dFormat = new SimpleDateFormat("dd-MMM-yyyy");
    String Date2 = dFormat.format(recordedDate_control.getDate());

 try{
    if((packetName.getSelectedIndex()==0 && Date.isEmpty()==true) ||
(packetName.getSelectedIndex()==0 || Date.isEmpty()==true)){
        notification.setText("Task Failed !!! ");
        if(packetName.getSelectedIndex()>0 && Date.isEmpty()==true)
        {
        JOptionPane.showMessageDialog(null,"Please Choose the Trace Record Date
!!","Error Alert !!!", JOptionPane.ERROR_MESSAGE);
        }

        if(packetName.getSelectedIndex()==0 && Date.isEmpty()==true)
        {
        JOptionPane.showMessageDialog(null,"Please select packet trace name and
Trace Record Date !!","Error Alert !!!", JOptionPane.ERROR_MESSAGE);
```

```
        }
        if(packetName.getSelectedIndex()==0 && Date.isEmpty()==false)
        {
        JOptionPane.showMessageDialog(null,"Please select packet trace name
!!","Error Alert !!!", JOptionPane.ERROR_MESSAGE);
        }
        }
      else{
      DefaultCategoryDataset detectionChart = new DefaultCategoryDataset();
      myQuery = conn.createStatement();
      myQueryx = conn.createStatement();
      rsQuery = myQuery.executeQuery("select * from attacks where
file_name='"+fileName+"' group by destination_ip");
        while(rsQuery.next())
        {
             Long IPd = rsQuery.getLong("destination_ip");
              String dest = ((IPd >> 24 ) & 0xFF) + "." + ((IPd >> 16 ) & 0xFF) + "." +
((IPd >> 8 ) & 0xFF) + "." +( IPd & 0xFF);

            rsQueryx = myQueryx.executeQuery("select * from attacks where
file_name='"+fileName+"' and destination_ip='"+IPd+"' and
recordedDate='"+Date2+"'");
            ArrayList<String> list= new ArrayList<String>();
             while(rsQueryx.next())
             {
              list.add(rsQueryx.getString("source_ip"));
             }
            detectionChart.setValue(list.size(), "Marks", dest);
            System.out.println(list.size()+" => "+dest);
         }
         JFreeChart chart = ChartFactory.createBarChart("SYN FLOODS ATTACKS
DETECTED RECORDED ON "+Date2, "DESTINATION IP
```

```
ADDRESSES(VICTIM)", "NUMBER OF ATTACKS", detectionChart,
PlotOrientation.VERTICAL, false, false, false);
            chart.setBackgroundPaint(Color.lightGray);
            chart.getTitle().setPaint(Color.red);
            chart.setBackgroundPaint(Color.WHITE);
            chart.setBorderPaint(Color.RED);
            CategoryPlot p = chart.getCategoryPlot();
            p.setRangeGridlinePaint(Color.BLUE);
            ChartFrame frame = new ChartFrame("Attacks in -> "+fileName,chart);
            frame.setVisible(true);
            frame.setSize(600, 600);
            frame.setBounds(400, 20, 600, 600);
            frame.setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE);



     }
  }catch (SQLException se) //catch block
  {
      //Handle errors for JDBC
     se.printStackTrace();
  } catch (Exception e) //catch block
  {
      //Handle errors for Class.forName
      e.printStackTrace();
  } finally //finally block
  {
      //finally block used to close resources
      try //try block
      {
         if (conn != null)//condition
         {
            conn.close(); //close connection
```

```
        }
      } catch (SQLException se)//Handle errors
      {
         se.printStackTrace();
      }//end finally try
   }//end try
   System.out.println("Goodbye!"); //print on console
}
```

APPENDIX H: FUNCTIONALITY CODE FOR "SYN FLOODS ATTACKS SUMMARY" SUBMENU

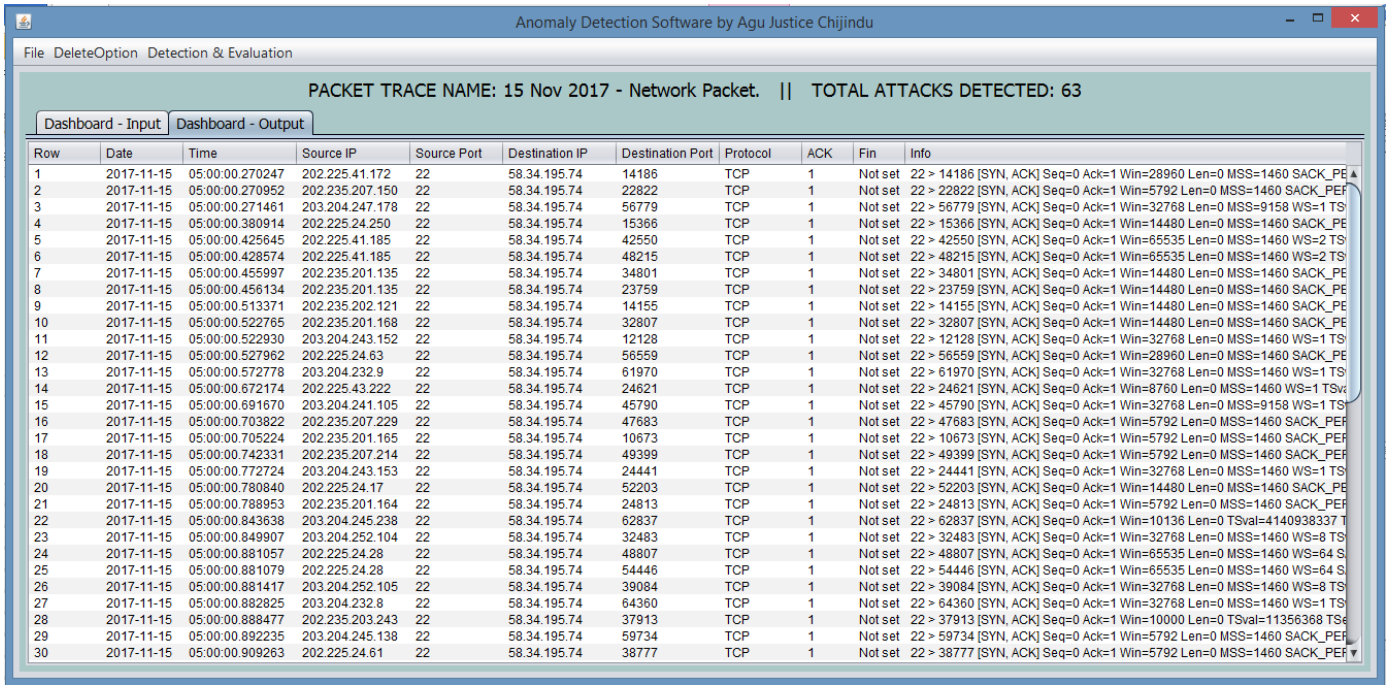
SYN Floods Attacks Summary

```
private void
SYN_FLoods_Attack_Summary_in_BarChartActionPerformed(java.awt.event.ActionE
vent evt) {
//Connection conn = mysqlConnector.ConnectDB();
Connection conn = dbconnection.ConnectDB();
   Statement myQuery = null;
   Statement myQueryx = null;
   ResultSet rsQuery = null;
   ResultSet rsQueryx = null;
   String Date =
((JTextField)recordedDate_control.getDateEditor().getUiComponent()).getText();
   SimpleDateFormat dFormat = new SimpleDateFormat("dd-MMM-yyyy");
    String Date2 = dFormat.format(recordedDate_control.getDate());

 try{
    if((packetName.getSelectedIndex()==0 && Date.isEmpty()==true) ||
(packetName.getSelectedIndex()==0 || Date.isEmpty()==true)){
          notification.setText("Task Failed !!! ");
          if(packetName.getSelectedIndex()>0 && Date.isEmpty()==true)
          {
          JOptionPane.showMessageDialog(null,"Please Choose the Trace Record Date
!!","Error Alert !!!", JOptionPane.ERROR_MESSAGE);
          }

          if(packetName.getSelectedIndex()==0 && Date.isEmpty()==true)
          {
          JOptionPane.showMessageDialog(null,"Please select packet trace name and
Trace Record Date !!","Error Alert !!!", JOptionPane.ERROR_MESSAGE);
```

```
        }
        if(packetName.getSelectedIndex()==0 && Date.isEmpty()==false)
        {
        JOptionPane.showMessageDialog(null,"Please select packet trace name
!!","Error Alert !!!", JOptionPane.ERROR_MESSAGE);
        }
        }
      else{
      DefaultCategoryDataset detectionChartSummary = new
DefaultCategoryDataset();
      myQuery = conn.createStatement();
      myQueryx = conn.createStatement();
      rsQuery = myQuery.executeQuery("select * from attacks group by file_name");
        while(rsQuery.next())
        {
             String fileName = rsQuery.getString("file_name");
            rsQueryx = myQueryx.executeQuery("select * from attacks where
file_name='"+fileName+"' and recordedDate='"+Date2+"'");
            ArrayList<String> list= new ArrayList<String>();
             while(rsQueryx.next())
             {
              list.add(rsQueryx.getString("source_ip"));
             }
            detectionChartSummary.setValue(list.size(), "Marks", fileName);
            System.out.println(list.size()+" => "+fileName);
         }
         JFreeChart chart = ChartFactory.createBarChart("SUMMARY OF SYN
FLOODS DDoS ATTACKS DETECTED RECORDED ON "+Date2, "PACKET
TRACE DATASETS", "NUMBER OF ATTACKS IN EACH PACKET TRACE
DATASETS", detectionChartSummary,  PlotOrientation.HORIZONTAL, false, false,
false);
          chart.setBackgroundPaint(Color.lightGray);
```

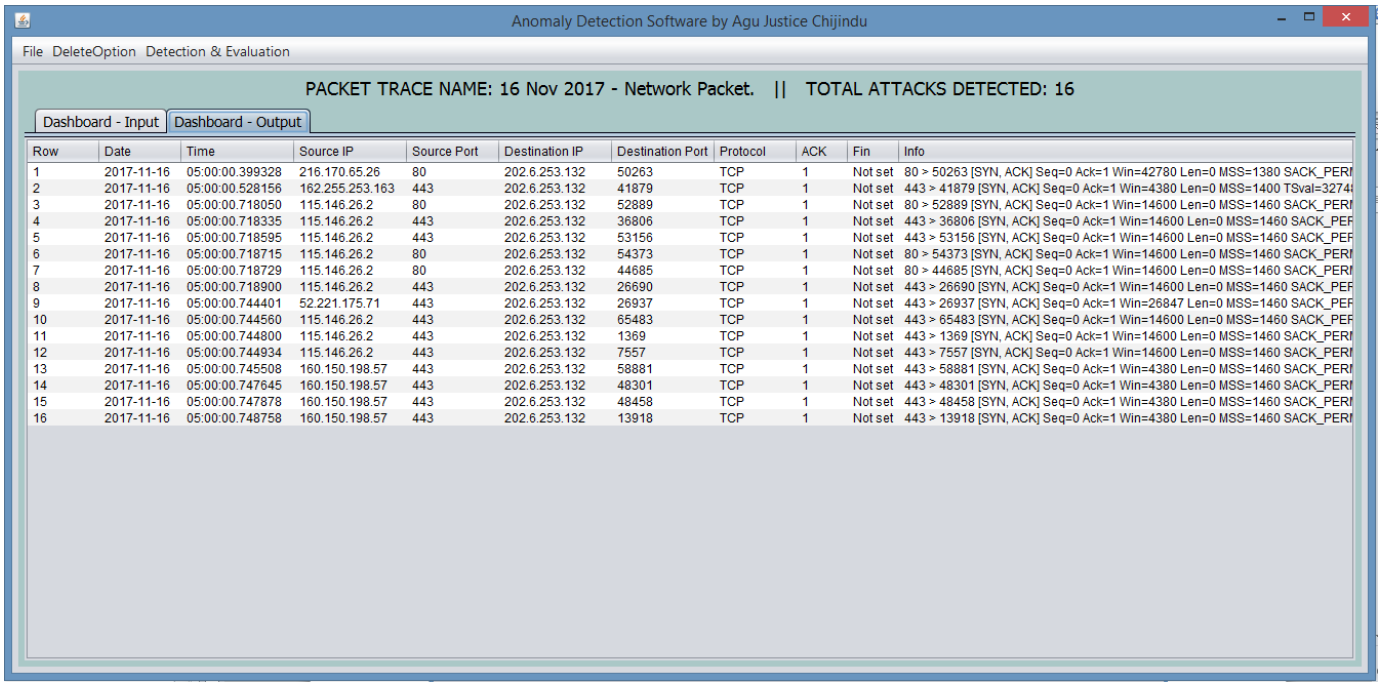
```
            chart.getTitle().setPaint(Color.red);
            chart.setBackgroundPaint(Color.WHITE);
            chart.setBorderPaint(Color.RED);
            CategoryPlot p = chart.getCategoryPlot();
            p.setRangeGridlinePaint(Color.BLUE);
            ChartFrame frame = new ChartFrame("Anormaly Detection Algorithm - Agu
Justice Chijindu",chart);
            frame.setVisible(true);
            frame.setSize(1300, 600);
            frame.setBounds(40, 45, 1300, 600);
     }
   }catch (SQLException se) //catch block
    {
       //Handle errors for JDBC
      se.printStackTrace();
    } catch (Exception e) //catch block
    {
       //Handle errors for Class.forName
       e.printStackTrace();
    } finally //finally block
    {
       //finally block used to close resources
       try //try block
       {
          if (conn != null)//condition
          {
             conn.close(); //close connection
          }
       } catch (SQLException se)//Handle errors
       { se.printStackTrace(); }//end finally try
}//end try
System.out.println("Goodbye!"); //print on console}
```

APPENDIX I: SYN FLOODS ATTACKS DETECTED IN "*15 Nov 2017 – Network Packet*" TRACE



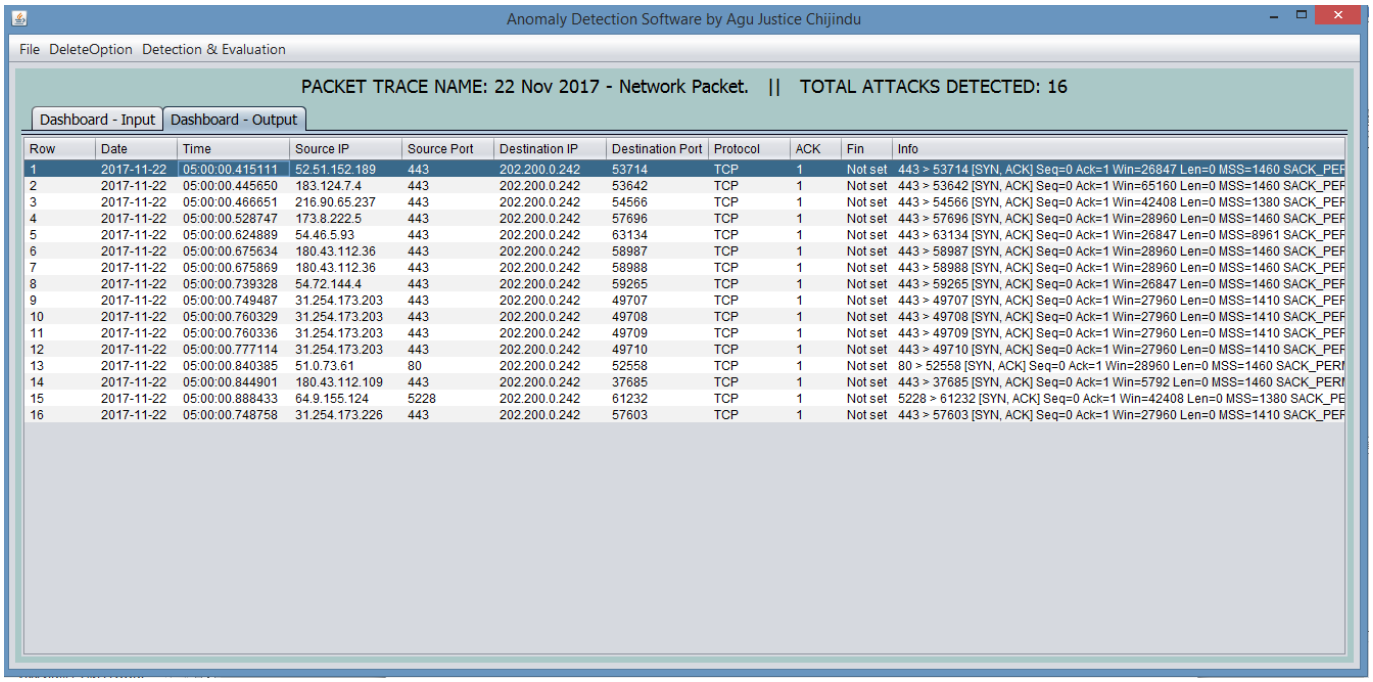
Row	Date	Time	Source IP	Source Port	Destination IP	Destination Port	Protocol	ACK	Fin	Info
1	2017-11-15	05:00:00.270247	202.225.41.172	22	58.34.195.74	14186	TCP	1	Not set	22 > 14186 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PE
2	2017-11-15	05:00:00.270952	202.235.207.150	22	58.34.195.74	22822	TCP	1	Not set	22 > 22822 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1460 SACK_PE
3	2017-11-15	05:00:00.271461	203.204.247.178	22	58.34.195.74	56779	TCP	1	Not set	22 > 56779 [SYN, ACK] Seq=0 Ack=1 Win=32768 Len=0 MSS=9158 WS=1 TS
4	2017-11-15	05:00:00.380914	202.225.24.250	22	58.34.195.74	15366	TCP	1	Not set	22 > 15366 [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1460 SACK_PE
5	2017-11-15	05:00:00.425645	202.225.41.185	22	58.34.195.74	42550	TCP	1	Not set	22 > 42550 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=2 TS
6	2017-11-15	05:00:00.428574	202.225.41.185	22	58.34.195.74	48215	TCP	1	Not set	22 > 48215 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=2 TS
7	2017-11-15	05:00:00.455997	202.235.201.135	22	58.34.195.74	34801	TCP	1	Not set	22 > 34801 [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1460 SACK_PE
8	2017-11-15	05:00:00.456134	202.235.201.135	22	58.34.195.74	23759	TCP	1	Not set	22 > 23759 [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1460 SACK_PE
9	2017-11-15	05:00:00.513371	202.235.202.121	22	58.34.195.74	14155	TCP	1	Not set	22 > 14155 [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1460 SACK_PE
10	2017-11-15	05:00:00.522765	202.235.201.168	22	58.34.195.74	32807	TCP	1	Not set	22 > 32807 [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1460 SACK_PE
11	2017-11-15	05:00:00.522930	203.204.243.152	22	58.34.195.74	12128	TCP	1	Not set	22 > 12128 [SYN, ACK] Seq=0 Ack=1 Win=32768 Len=0 MSS=1460 WS=1 TS
12	2017-11-15	05:00:00.527962	202.225.24.63	22	58.34.195.74	56559	TCP	1	Not set	22 > 56559 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PE
13	2017-11-15	05:00:00.572778	203.204.232.9	22	58.34.195.74	61970	TCP	1	Not set	22 > 61970 [SYN, ACK] Seq=0 Ack=1 Win=32768 Len=0 MSS=1460 WS=1 TS
14	2017-11-15	05:00:00.672174	202.225.43.222	22	58.34.195.74	24621	TCP	1	Not set	22 > 24621 [SYN, ACK] Seq=0 Ack=1 Win=8760 Len=0 MSS=1460 WS=1 TSv
15	2017-11-15	05:00:00.691670	203.204.241.105	22	58.34.195.74	45790	TCP	1	Not set	22 > 45790 [SYN, ACK] Seq=0 Ack=1 Win=32768 Len=0 MSS=9158 WS=1 TS
16	2017-11-15	05:00:00.703822	202.235.207.229	22	58.34.195.74	47683	TCP	1	Not set	22 > 47683 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1460 SACK_PE
17	2017-11-15	05:00:00.705224	202.235.201.165	22	58.34.195.74	10673	TCP	1	Not set	22 > 10673 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1460 SACK_PE
18	2017-11-15	05:00:00.742331	202.235.207.214	22	58.34.195.74	49399	TCP	1	Not set	22 > 49399 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1460 SACK_PE
19	2017-11-15	05:00:00.772724	203.204.243.153	22	58.34.195.74	24441	TCP	1	Not set	22 > 24441 [SYN, ACK] Seq=0 Ack=1 Win=32768 Len=0 MSS=1460 WS=1 TS
20	2017-11-15	05:00:00.780840	202.225.24.17	22	58.34.195.74	52203	TCP	1	Not set	22 > 52203 [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1460 SACK_PE
21	2017-11-15	05:00:00.788953	202.235.201.164	22	58.34.195.74	24813	TCP	1	Not set	22 > 24813 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1460 SACK_PE
22	2017-11-15	05:00:00.843638	203.204.245.238	22	58.34.195.74	62837	TCP	1	Not set	22 > 62837 [SYN, ACK] Seq=0 Ack=1 Win=10136 Len=0 TSval=4140938337 T
23	2017-11-15	05:00:00.849907	203.204.252.104	22	58.34.195.74	32483	TCP	1	Not set	22 > 32483 [SYN, ACK] Seq=0 Ack=1 Win=32768 Len=0 MSS=1460 WS=8 TS
24	2017-11-15	05:00:00.881057	202.225.24.28	22	58.34.195.74	48807	TCP	1	Not set	22 > 48807 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=64 S
25	2017-11-15	05:00:00.881079	202.225.24.28	22	58.34.195.74	54446	TCP	1	Not set	22 > 54446 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=64 S
26	2017-11-15	05:00:00.881417	203.204.252.105	22	58.34.195.74	39084	TCP	1	Not set	22 > 39084 [SYN, ACK] Seq=0 Ack=1 Win=32768 Len=0 MSS=1460 WS=8 TS
27	2017-11-15	05:00:00.882825	203.204.232.8	22	58.34.195.74	64360	TCP	1	Not set	22 > 64360 [SYN, ACK] Seq=0 Ack=1 Win=32768 Len=0 MSS=1460 WS=1 TS
28	2017-11-15	05:00:00.888477	202.235.203.243	22	58.34.195.74	37913	TCP	1	Not set	22 > 37913 [SYN, ACK] Seq=0 Ack=1 Win=10000 Len=0 TSval=11356368 TSe
29	2017-11-15	05:00:00.892235	203.204.245.138	22	58.34.195.74	59734	TCP	1	Not set	22 > 59734 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1460 SACK_PE
30	2017-11-15	05:00:00.909263	202.225.24.61	22	58.34.195.74	38777	TCP	1	Not set	22 > 38777 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1460 SACK_PE

APPENDIX J: SYN FLOODS ATTACKS DETECTED IN "*16 Nov 2017 – Network Packet*" TRACE



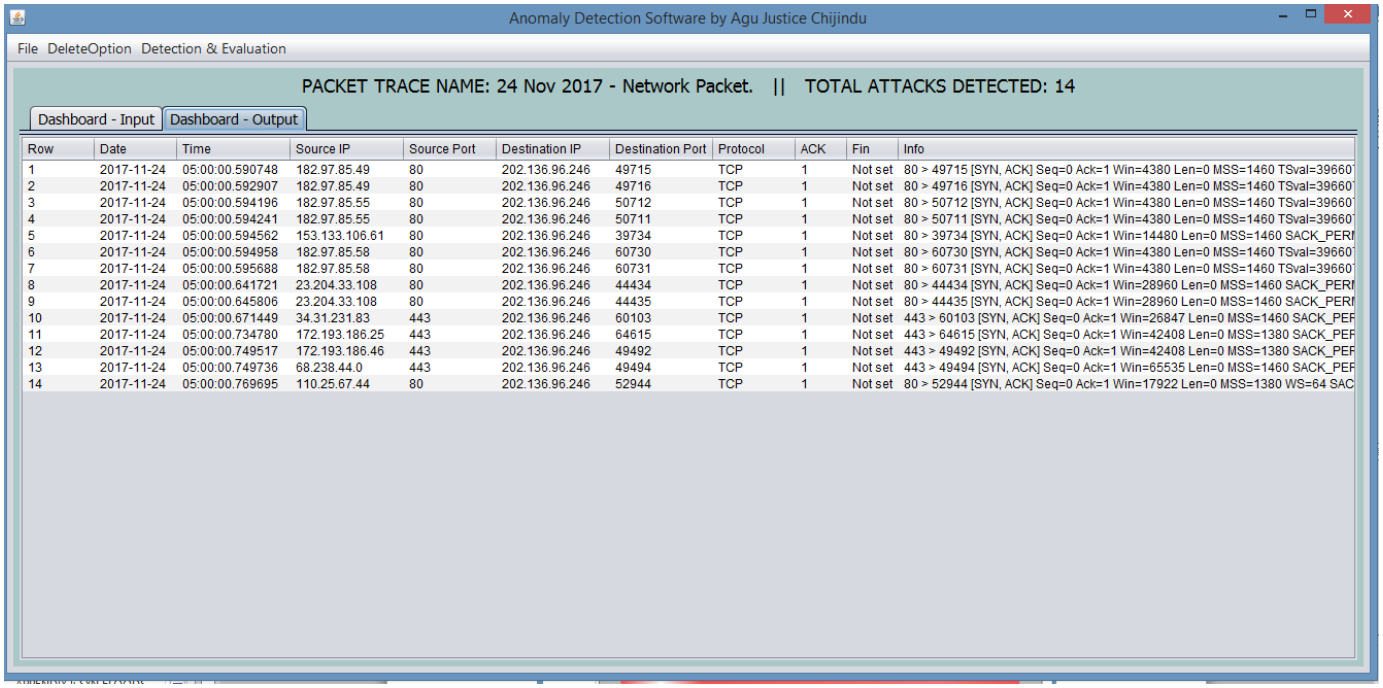
Row	Date	Time	Source IP	Source Port	Destination IP	Destination Port	Protocol	ACK	Fin	Info
1	2017-11-16	05:00:00.399328	216.170.65.26	80	202.6.253.132	50263	TCP	1	Not set	80 > 50263 [SYN, ACK] Seq=0 Ack=1 Win=42780 Len=0 MSS=1380 SACK_PERI
2	2017-11-16	05:00:00.528156	162.255.253.163	443	202.6.253.132	41879	TCP	1	Not set	443 > 41879 [SYN, ACK] Seq=0 Ack=1 Win=4380 Len=0 MSS=1400 TSval=3274
3	2017-11-16	05:00:00.718050	115.146.26.2	80	202.6.253.132	52889	TCP	1	Not set	80 > 52889 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0 MSS=1460 SACK_PERI
4	2017-11-16	05:00:00.718335	115.146.26.2	443	202.6.253.132	36806	TCP	1	Not set	443 > 36806 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0 MSS=1460 SACK_PER
5	2017-11-16	05:00:00.718595	115.146.26.2	443	202.6.253.132	53156	TCP	1	Not set	443 > 53156 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0 MSS=1460 SACK_PER
6	2017-11-16	05:00:00.718715	115.146.26.2	80	202.6.253.132	54373	TCP	1	Not set	80 > 54373 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0 MSS=1460 SACK_PERI
7	2017-11-16	05:00:00.718729	115.146.26.2	80	202.6.253.132	44685	TCP	1	Not set	80 > 44685 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0 MSS=1460 SACK_PERI
8	2017-11-16	05:00:00.718900	115.146.26.2	443	202.6.253.132	26690	TCP	1	Not set	443 > 26690 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0 MSS=1460 SACK_PER
9	2017-11-16	05:00:00.744401	52.221.175.71	443	202.6.253.132	26937	TCP	1	Not set	443 > 26937 [SYN, ACK] Seq=0 Ack=1 Win=26847 Len=0 MSS=1460 SACK_PER
10	2017-11-16	05:00:00.744560	115.146.26.2	443	202.6.253.132	65483	TCP	1	Not set	443 > 65483 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0 MSS=1460 SACK_PER
11	2017-11-16	05:00:00.744800	115.146.26.2	443	202.6.253.132	1369	TCP	1	Not set	443 > 1369 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0 MSS=1460 SACK_PERI
12	2017-11-16	05:00:00.744934	115.146.26.2	443	202.6.253.132	7557	TCP	1	Not set	443 > 7557 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0 MSS=1460 SACK_PERI
13	2017-11-16	05:00:00.745508	160.150.198.57	443	202.6.253.132	58881	TCP	1	Not set	443 > 58881 [SYN, ACK] Seq=0 Ack=1 Win=4380 Len=0 MSS=1460 SACK_PERI
14	2017-11-16	05:00:00.747645	160.150.198.57	443	202.6.253.132	48301	TCP	1	Not set	443 > 48301 [SYN, ACK] Seq=0 Ack=1 Win=4380 Len=0 MSS=1460 SACK_PERI
15	2017-11-16	05:00:00.747878	160.150.198.57	443	202.6.253.132	48458	TCP	1	Not set	443 > 48458 [SYN, ACK] Seq=0 Ack=1 Win=4380 Len=0 MSS=1460 SACK_PERI
16	2017-11-16	05:00:00.748758	160.150.198.57	443	202.6.253.132	13918	TCP	1	Not set	443 > 13918 [SYN, ACK] Seq=0 Ack=1 Win=4380 Len=0 MSS=1460 SACK_PERI

APPENDIX K: SYN FLOODS ATTACKS DETECTED IN "*22 Nov 2017 – Network Packet*" TRACE



Row	Date	Time	Source IP	Source Port	Destination IP	Destination Port	Protocol	ACK	Fin	Info
1	2017-11-22	05:00:00.415111	52.51.152.189	443	202.200.0.242	53714	TCP	1	Not set	443 > 53714 [SYN, ACK] Seq=0 Ack=1 Win=26847 Len=0 MSS=1460 SACK_PER
2	2017-11-22	05:00:00.445650	183.124.7.4	443	202.200.0.242	53642	TCP	1	Not set	443 > 53642 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PER
3	2017-11-22	05:00:00.466651	216.90.65.237	443	202.200.0.242	54566	TCP	1	Not set	443 > 54566 [SYN, ACK] Seq=0 Ack=1 Win=42408 Len=0 MSS=1380 SACK_PER
4	2017-11-22	05:00:00.528747	173.8.222.5	443	202.200.0.242	57696	TCP	1	Not set	443 > 57696 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PER
5	2017-11-22	05:00:00.624889	54.46.5.93	443	202.200.0.242	63134	TCP	1	Not set	443 > 63134 [SYN, ACK] Seq=0 Ack=1 Win=26847 Len=0 MSS=8961 SACK_PER
6	2017-11-22	05:00:00.675634	180.43.112.36	443	202.200.0.242	58987	TCP	1	Not set	443 > 58987 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PER
7	2017-11-22	05:00:00.675869	180.43.112.36	443	202.200.0.242	58988	TCP	1	Not set	443 > 58988 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PER
8	2017-11-22	05:00:00.739328	54.72.144.4	443	202.200.0.242	59265	TCP	1	Not set	443 > 59265 [SYN, ACK] Seq=0 Ack=1 Win=26847 Len=0 MSS=1460 SACK_PER
9	2017-11-22	05:00:00.749487	31.254.173.203	443	202.200.0.242	49707	TCP	1	Not set	443 > 49707 [SYN, ACK] Seq=0 Ack=1 Win=27960 Len=0 MSS=1410 SACK_PER
10	2017-11-22	05:00:00.760329	31.254.173.203	443	202.200.0.242	49708	TCP	1	Not set	443 > 49708 [SYN, ACK] Seq=0 Ack=1 Win=27960 Len=0 MSS=1410 SACK_PER
11	2017-11-22	05:00:00.760336	31.254.173.203	443	202.200.0.242	49709	TCP	1	Not set	443 > 49709 [SYN, ACK] Seq=0 Ack=1 Win=27960 Len=0 MSS=1410 SACK_PER
12	2017-11-22	05:00:00.777114	31.254.173.203	443	202.200.0.242	49710	TCP	1	Not set	443 > 49710 [SYN, ACK] Seq=0 Ack=1 Win=27960 Len=0 MSS=1410 SACK_PER
13	2017-11-22	05:00:00.840385	51.0.73.61	80	202.200.0.242	52558	TCP	1	Not set	80 > 52558 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERI
14	2017-11-22	05:00:00.844901	180.43.112.109	443	202.200.0.242	37685	TCP	1	Not set	443 > 37685 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1460 SACK_PERI
15	2017-11-22	05:00:00.888433	64.9.155.124	5228	202.200.0.242	61232	TCP	1	Not set	5228 > 61232 [SYN, ACK] Seq=0 Ack=1 Win=42408 Len=0 MSS=1380 SACK_PE
16	2017-11-22	05:00:00.748758	31.254.173.226	443	202.200.0.242	57603	TCP	1	Not set	443 > 57603 [SYN, ACK] Seq=0 Ack=1 Win=27960 Len=0 MSS=1410 SACK_PER

APPENDIX M: SYN FLOODS ATTACKS DETECTED IN "*24 Nov 2017 – Network Packet*" TRACE



Row	Date	Time	Source IP	Source Port	Destination IP	Destination Port	Protocol	ACK	Fin	Info
1	2017-11-24	05:00:00.590748	182.97.85.49	80	202.136.96.246	49715	TCP	1	Not set	80 > 49715 [SYN, ACK] Seq=0 Ack=1 Win=4380 Len=0 MSS=1460 TSval=39660
2	2017-11-24	05:00:00.592907	182.97.85.49	80	202.136.96.246	49716	TCP	1	Not set	80 > 49716 [SYN, ACK] Seq=0 Ack=1 Win=4380 Len=0 MSS=1460 TSval=39660
3	2017-11-24	05:00:00.594196	182.97.85.55	80	202.136.96.246	50712	TCP	1	Not set	80 > 50712 [SYN, ACK] Seq=0 Ack=1 Win=4380 Len=0 MSS=1460 TSval=39660
4	2017-11-24	05:00:00.594241	182.97.85.55	80	202.136.96.246	50711	TCP	1	Not set	80 > 50711 [SYN, ACK] Seq=0 Ack=1 Win=4380 Len=0 MSS=1460 TSval=39660
5	2017-11-24	05:00:00.594562	153.133.106.61	80	202.136.96.246	39734	TCP	1	Not set	80 > 39734 [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1460 SACK_PERI
6	2017-11-24	05:00:00.594958	182.97.85.58	80	202.136.96.246	60730	TCP	1	Not set	80 > 60730 [SYN, ACK] Seq=0 Ack=1 Win=4380 Len=0 MSS=1460 TSval=39660
7	2017-11-24	05:00:00.595688	182.97.85.58	80	202.136.96.246	60731	TCP	1	Not set	80 > 60731 [SYN, ACK] Seq=0 Ack=1 Win=4380 Len=0 MSS=1460 TSval=39660
8	2017-11-24	05:00:00.641721	23.204.33.108	80	202.136.96.246	44434	TCP	1	Not set	80 > 44434 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERI
9	2017-11-24	05:00:00.645806	23.204.33.108	80	202.136.96.246	44435	TCP	1	Not set	80 > 44435 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERI
10	2017-11-24	05:00:00.671449	34.31.231.83	443	202.136.96.246	60103	TCP	1	Not set	443 > 60103 [SYN, ACK] Seq=0 Ack=1 Win=26847 Len=0 MSS=1460 SACK_PER
11	2017-11-24	05:00:00.734780	172.193.186.25	443	202.136.96.246	64615	TCP	1	Not set	443 > 64615 [SYN, ACK] Seq=0 Ack=1 Win=42408 Len=0 MSS=1380 SACK_PER
12	2017-11-24	05:00:00.749517	172.193.186.46	443	202.136.96.246	49492	TCP	1	Not set	443 > 49492 [SYN, ACK] Seq=0 Ack=1 Win=42408 Len=0 MSS=1380 SACK_PER
13	2017-11-24	05:00:00.749736	68.238.44.0	443	202.136.96.246	49494	TCP	1	Not set	443 > 49494 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 SACK_PER
14	2017-11-24	05:00:00.769695	110.25.67.44	80	202.136.96.246	52944	TCP	1	Not set	80 > 52944 [SYN, ACK] Seq=0 Ack=1 Win=17922 Len=0 MSS=1380 WS=64 SAC