NETWORK ANOMALY DETECTION ENGINE

(NADE)

By

Maryam Shafique

Mazhar Abbas

Arsalan Aslam

Supervisor

Asst. Prof Waleed Bin Shahid

Submitted to the faculty of Department of Software Engineering,

Military College of Signals, National University of Sciences and Technology,

in partial fulfillment for the requirements of B.E Degree in Software Engineering

JULY 2021

CERTIFICATE OF CORRECTIONS & APPROVAL

Certified that work contained in this thesis titled "Network Anomaly Detection Engine (NADE)",carried out by Maryam Shafique , Mazhar Abbas and Arsalan Aslam under the supervision of Asst. Prof. Waleed Bin Shahid for partial fulfillment of Degree of Bachelor of Software Engineering, in Military College of Signals, National University of Sciences and Technology, Islamabad during the academic year 2021 is correct and approved. The material that has been used from other sources it has been properly acknowledged / referred.

Approved by

Signature:

Asst. Prof. Waleed Bin Shahid

(Supervisor)

Signature:

Assoc Prof. Dr. Hammad Afzal

(Co-Supervisor)

Date: 12 July 2021

DECLARATION

No portion of work presented in this thesis has been submitted in support of another award or qualification in either this institute or anywhere else.

PLAGIARISM CERTIFICATE (TURNITIN REPORT)

This thesis has been checked for Plagiarism. Turnitin report endorsed by Supervisor is attached.

Maryam Shafique

Regn No: 00000240974

Signature:

Mazhar Abbas

Regn No: 00000240970

Signature:

Arslan Aslam

Regn No: 00000240957

Signature:

Signature of Supervisor

ACKNOWLEDGEMENTS

We are thankful to the Creator Allah Subhana-Watala to have guided us throughout this work at every step and for every new thought which You setup in our mind to improve it. Indeed, we could have done nothing without Your priceless help and guidance. Whosoever helped us throughout the course of our thesis, whether our parents or any other individual was Your will, so indeed none be worthy of praise but You.

We are profusely thankful to our beloved parents who raised us when we were not capable of walking and continued to support us throughout in every department of our life.

We would also like to express special thanks to our supervisor Asst. Prof. Waleed Bin Shahid for his help throughout the thesis and for their consistent direction and inspiration throughout our venture. Without their assistance, we would not have the option to achieve anything.

Finally, we would like to express our gratitude to all the individuals who have rendered valuable assistance to our study.

Dedicated to our exceptional parents and adored siblings whose tremendous support and cooperation led us to this wonderful accomplishment.

Abstract

IT environments are growing ever more distributed, complex, and difficult to manage whereas cyber-attacks are becoming more and more common. Attackers constantly look to exploit any gap in IT systems, applications, and hardware to compromise confidentiality, integrity, and availability of information. With rapidly increasing cyber-attacks, the old preventative, and defensive techniques of simply using firewalls, antivirus software and conventional IDS stand incapacitated to detect advanced network attacks. This accentuates the need to come up with an elaborate NextGen Network Anomaly Detection Engine which monitors the attack and threat landscape in real-time using advanced techniques.

A Network Anomaly Detection Engine can detect advanced network attacks in real-time with the help of Machine Learning techniques. It would improve security visibility and actionability along with an in-depth analysis of incoming and outgoing traffic. NADE will use custom Zeek[1] scripts to extract useful features from network traffic that will include both attack and benign network data. Then NADE will use Machine Learning driven techniques to detect advanced threats which includes scanning, DoS attacks and other Network layer attacks. Moreover, our solution, the Network Anomaly Detection Engine (NADE) will provide a platform where all logs are gathered, and unusual behavior is detected.

Key Words: *NADE, Machine Learning, Network Attacks*

Table of Contents

CHAPTER 1: INTRODUCTION ... 1
1.1 Overview ... 1
1.2 Scope ... 1
1.3 Product Functions ... 1
1.4 Deliverables ... 2
1.5 Overview of the Document ... 2
1.5.1 Document Conventions ... 3
1.5.2 Headings ... 3
1.5.3 References ... 3
1.5.4 Basic Text ... 3
CHAPTER 2: LITERATURE REVIEW ... 4
CHAPTER 3 : METHODOLOGY ... 6
3.1 Phase 1 ... 6
3.2 Phase 2 ... 6
3.2.1 DATASET ... 6
3.2.2 CICFlowmeter ... 7
3.2.3 ZeekFlowmeter ... 7
3.2.4 Characteristics of the Dataset: ... 14
3.2.5 Dataset Description ... 14
3.2.6 Dataset Analysis ... 15
3.2.7 ATTACKS: ... 16
3.2.8 MACHINE LEARNING: ... 21
3.2.9 RESULTS: ... 23
3.2.10 Classifiers Performance ... 25
3.3 Phase 3 ... 31
3.3.1 Elastic Search ... 31
3.3.2 Kibana: ... 31
CHAPTER 4 : SOFTWARE REQUIREMENT SPECIFICATION ... 35
4.1 Introduction ... 35
4.1.1 Purpose ... 35
4.2 System Overview ... 35
4.2.1 Product Perspective ... 35
4.2.2 User Classes and Characteristics ... 36
4.2.3 Operating Environment ... 36
4.2.4 Design and Implementation Constraints ... 37
4.2.5 User Documentation ... 37
4.2.6 Assumptions and Dependencies ... 37

4.3 External Interface Requirements ... 38
4.3.1 Hardware Interfaces ... 38
4.3.2 Software Interfaces ... 38
4.3.3 Communications Interfaces ... 38
4.4 System Features ... 38
4.5 Overall Use Case Diagram ... 39
4.5.1 Authentication ... 39
4.5.2 Capture Live Network Traffic ... 41
4.5.3 Feature Extraction through Zeek ... 42
4.5.4 Anomaly Detection using Machine Learning ... 43
4.5.5 Data Indexing ... 44
4.5.6 Data Visualization ... 46
4.6 Other Non-Functional Requirements ... 47
4.6.1 Performance Requirements ... 47
4.6.2 Safety Requirements ... 47
4.6.3 Security Requirements ... 47
4.6.4 Software Quality Attributes ... 47
4.6.5 Business Rules ... 48
CHAPTER 5 : DESIGN AND DEVELOPMENT ... 49
5.1 Introduction ... 49
5.1.1 Purpose ... 49
5.2 System architecture ... 49
5.2.1 Architectural Design ... 49
5.2.2 Decomposition Description ... 50
5.2.3 Design Rationale ... 55
5.3 Data Design ... 56
5.3.1 Data Description ... 56
5.3.2 Data Dictionary ... 56
CHAPTER 6 : TESTING ... 57
6.1 TEST CASE # 1 ... 57
6.2 TEST CASE # 2 ... 59
6.3 TEST CASE # 3 ... 61
6.4 TEST CASE # 4 ... 62
6.5 TEST CASE # 5 ... 64
6.6 TEST CASE # 6 ... 65
FUTURE WORK ... 67
REFERENCES: ... 67

CHAPTER 1: INTRODUCTION

1.1 Overview

The size and complexity of today's enterprises is growing exponentially, along with the number of IT personnel to support them. This makes information sharing and collaboration difficult when problems occur. NADE allows security teams to keep on top of security alerts in real-time. By gathering events across the network, a NADE can determine the nature of attack. The main goal of NADE is to improve security, visibility and actionability along with an in-depth analysis of incoming and outgoing traffic.

Network Anomaly Detection Engine (NADE) will log the actions that users take on the network, create a baseline to train the ML-driven model which will eventually facilitate the detection, and ultimate halting of network attacks. Our solution, the Network Anomaly Detection Engine (NADE) will provide a platform where all logs are gathered, and unusual behavior is detected, and attacks are visualized which would add value to the overall security posture of the organization where it is applied.

1.2 Scope

The scope of NADE is to provide a cost-effective yet comprehensive solution that will benefit organizations and individuals in protecting their data against loss or cyber theft. Next-Generation Network Anomaly Detection Engine will capture live traffic and transform it into a format where useful features can be seen and evaluated using Zeek scripts. Then this transformed traffic is directed to the ML model for anomaly detection. If any anomaly is detected, it will be indexed using ELK Stack and visualized on Kibana Dashboard. It will improve security visibility, actionability, and posture while reducing analysts' burden. It will also prevent noise/false-positive results by using advanced Machine Learning techniques.

1.3 Product Functions

The main functions of NADE are highlighted below:

- Capture live network traffic and extract useful features using Zeek scripts.
- Analyze and detect abnormal or suspicious user behavior, advanced threats and security breaches in network using trained Machine Learning Model.
- Index the data for efficient searching and presenting the organized data for end-user.
- Generate operational security dashboard and reports to have a full visibility of security attacks for Network Administrator.

1.4 Deliverables

Sr.	Tasks	Deliverables
1	Literature Review	Literature Survey
2	Requirements Gathering	SRS Document
3	Application Design	Design Document (SDS)
4	Implementation	Implementation on computer with a live test to show the accuracy and ability of the project
5	Testing	Evaluation plan and test document
6	Training	Deployment Plan
7	Deployment	Complete application along with necessary documentation

1.5 Overview of the Document

This document shows the complete working process of our project NADE. It starts with the literature review which shows past work done in a similar field, requirement analysis of the system, system architecture which highlights the modules of the software and represents the system in the form of a component diagram, Use Case Diagram, Sequence Diagram, and general design of the system. Then it will move on to discuss the detailed Description of all the components involved. Further, the dependencies of the system and its relationship with other products and the capacity of it to be reused will be discussed.

1.5.1 Document Conventions

This section describes the standards followed while writing the document.

1.5.2 Headings

Headings are prioritized in a numbered fashion, the highest priority heading having a single digit and subsequent headings having more numbers, per their level. All the main headings are titled as follows: single-digit number followed by a dot and the name of the section (All bold Times New Roman, size 18, Centered).

All second-level subheadings for every subsection have the same number as their respective main heading, followed by one dot and subsequent subheading number followed by name of the subsection (All bold Times New Roman, size 16). Further subheadings, i.e., level three and below, follow the same rules as above for numbering and naming, but different for the font (All bold Times New Roman, size 14).

1.5.3 References

All references in this document are provided where necessary, however, were not present, the meaning is self-explanatory. All ambiguous terms have been clarified in the glossary at the end of this document.

1.5.4 Basic Text

All other basic text appears in regular, size 12 Times New Roman. Every paragraph explains one type of idea.

CHAPTER 2: LITERATURE REVIEW

CIC dataset was created by Sharafaldin et al, Ashkari et al, and Ghorbani et al [3]. They proposed a technique towards generating a new I DS dataset. They analyzed different IDS dataset and proposed their approach to generate dataset for IDS. They extracted 84 features from network traffic. They also discussed their environment configurations that was used to generate dataset. The focus of their approach is to make a dataset having features for detection by machine learning.

We explored dataset thoroughly and, in this context, read some papers on CIC dataset analysis [2]. In [2] they discussed shortcomings in the dataset. Although this is a state-of-the-art dataset [3] and created by a well-known Institution of cyber security but it has some shortcoming that are discussed in detail [2].

[2] Problem with dataset is this that it is highly class imbalanced which made us to drop some rare-occurring attack traffic unfortunately. Second shortcoming is this that size of dataset is so large that it cannot be processed on limited resource systems.

A similar approach to our approach is used in [1] to detect malicious traffic with the help of Machine Learning. They extracted features from network traffic using customized Zeek scripts and trained their model on CIC dataset after converting into csv format. They deployed model in offline environment and measured its performance. But their approach is not capable of detecting real-time attacks and does not provide a GUI (Graphical User Interface) for the visualization of attacks.

Ahmim et al. [4] proposed a novel intrusion detection system (IDS) that combines different classifier approaches which are based on decision tree and rules-based concepts, namely, REP Tree, JRip algorithm and Forest PA. Specifically, the first and second method take as inputs features of the dataset and classify the network traffic as Attack/Benign. The third classifier uses features of the initial data set in addition to the outputs of the first and the second classifier as inputs. The experimental results obtained by analyzing the proposed IDS using the CICIDS2017 dataset, attest their superiority in terms of accuracy, detection rate, false alarm rate and time overhead as compared to state of the art existing schemes.

There are many techniques have been proposed to detect malicious network traffic but most of them lack in machine learning and graphical dashboard. We took intuition form [1]. Our approach may be considered as a future work of [1]. We used advanced Zeek scripts to extract network traffic features. We have taken [1] to next level by developing a module for online detection of attacks and GUI for graphical Interface.

We took data sampling technique from Ahmin et al [4]. They included a large proportion of benign traffic in dataset as most of the time there will be benign traffic in the network. In their novel hierarchical approach, they used two classifiers to detect attack traffic. First classifier tells whether incoming traffic is benign or attack. Second classifier predicts the type of attack if first classifier has predicted incoming traffic as attack. We experimented this approach, but results were not improving by using hierarchical approach. So, we decided to use one classifier for the detection of attack class.

CHAPTER 3 : METHODOLOGY

We are intended to develop an Intrusion Detection System which has capability to detect network layer attacks with minimum false alarms and an interactive graphical dashboard which can visualize detected attacks in network traffic.
Our proposed approach has three phases.

3.1 Phase 1:

Zeek is a an open-source network monitoring tool, and it has its own scripting language. We used Zeek scripting language to sniff network traffic. The Zeek script [7] extracts 84 features from network traffic. These features include CIC dataset features and some additional features that are useful for attack detection.

Zeek writes extracted features in flowmeter.log file. We made a parameter named 'duration' which sets the time for which the network traffic will be captured by the Zeek. After the traffic has been captured and written in flowmeter.log file, flowmeter.log file is accessed and is converted into pandas data frame.

3.2 Phase 2:

In second phase, detection of attack is done using Machine Learning. As we have been able to convert network traffic into pandas data frame , now we can pass it to trained machine learning model which classifies whether incoming traffic is benign or some sort of attack.

3.2.1 DATASET:

We used CICIDS2017 dataset [5] to train the model. CICIDS2017 dataset contains benign and the most up-to-date common attacks, which resembles the true real-world data (PCAPs). It also includes the results of the network traffic analysis using CICFlowMeter with labeled flows based on the time stamp, source, and destination IPs, source and destination ports, protocols, and attack (CSV files).

They made a CICFlowmeter [6] tool to convert pcap files to csv format to analyze the dataset. But their tool is not capable of working in real-time scenario. This is the reason we had to use Zeek scripting language to extract required features.

3.2.2 CICFlowmeter:

CICFlowMeter is a network traffic flow generator and analyzer. It can be used to generate bidirectional flows, where the first packet determines the forward (source to destination) and backward (destination to source) directions, hence more than 80 statistical network traffic features such as Duration, Number of packets, Number of bytes, Length of packets, etc. can be calculated separately in the forward and backward directions.

Additional functionalities include, selecting features from the list of existing features, adding new features, and controlling the duration of flow timeout. The output of the application is the CSV format file that has six columns labeled for each flow (flow_id, src_ip, dst_ip src_port, dst_port, and protocol) with more than 80 network traffic analysis features.

TCP flows are usually terminated upon connection teardown (by FIN packet) while UDP flows are terminated by a flow timeout. The flow timeout value can be assigned arbitrarily by the individual scheme e.g., 600 seconds for both TCP and UDP.

3.2.3 ZeekFlowmeter:

Zeek flowmeter is a tool (or script) written zeek scripting language which extracts CIC Dataset features from network traffic. This tool can extract features from both pcap files and live traffic.

Flowmeter performs layer 3 and 4 network traffic analysis and generates a set of new features based on timing, volume, and metadata. These features are ideal for developing models for traffic classification without using deep packet inspection. The extracted features are as follows:

Feature Name	Description	exists in CICFlowMeter
uid	The ID of the flow as given by Zeek	No
flow_duration	The length of the flow in seconds (maximal precision ms). If only on packet was seen the duration is 0.	Yes

fwd_pkts_tot	The number of packets travelling in the forward direction.	Yes
bwd_pkts_tot	The number of packets travelling in the backwards direction.	Yes
fwd_data_pkts_tot	The number of packets travelling in the forward direction, which have a payload.	Yes
bwd_data_pkts_tot	The number of packets travelling in the backwards direction, which have a payload.	No
fwd_pkts_per_sec	The average number of forward packets transmitted per second during the flow. If the duration is 0 then this feature is also set to 0.	Yes
bwd_pkts_per_sec	The average number of backward packets transmitted per second during the flow. If the duration is 0 then this feature is also set to 0.	Yes
flow_pkts_per_sec	The average number of packets transmitted per second during the flow. If the duration is 0 then this feature is also set to 0.	Yes
down_up_ratio	The number of backward packets divided by the number of forward packets. If the number of forward packets is 0 this feature is also set to 0.	Yes
fwd_header_size_tot	The total number of bytes the headers of the forward packets contained.	Yes
fwd_header_size_min	The number of bytes the smallest headers of the forward packets contained.	Yes
fwd_header_size_max	The number of bytes the largest headers of the forward packets contained.	Yes
bwd_header_size_tot	The total number of bytes the headers of the backward packets contained.	Yes
bwd_header_size_min	The number of bytes the smallest headers of the backward packets contained.	No

bwd_header_size_max	The number of bytes the largest headers of the backward packets contained.	No
fwd_pkts_payload.max	The largest payload size, in bytes, seen in the forward direction.	Yes
fwd_pkts_payload.min	The smallest payload size, in bytes, seen in the forward direction.	Yes
fwd_pkts_payload.tot	The total payload size, in bytes, seen in the forward direction.	Yes
fwd_pkts_payload.avg	The average payload size, in bytes, seen in the forward direction.	Yes
fwd_pkts_payload.std	The standard deviation of the payload size, in bytes, seen in the forward direction.	Yes
bwd_pkts_payload.max	The largest payload size, in bytes, seen in the backward direction.	Yes
bwd_pkts_payload.min	The smallest payload size, in bytes, seen in the backward direction.	Yes
bwd_pkts_payload.tot	The total payload size, in bytes, seen in the backward direction.	Yes
bwd_pkts_payload.avg	The average payload size, in bytes, seen in the backward direction.	Yes
bwd_pkts_payload.std	The standard deviation of the payload size, in bytes, seen in the backward direction.	Yes
flow_pkts_payload.max	The largest payload size, in bytes, seen in the flow.	Yes
flow_pkts_payload.min	The smallest payload size, in bytes, seen in the flow.	Yes
flow_pkts_payload.tot	The total payload size, in bytes, seen in the flow.	No
flow_pkts_payload.avg	The average payload size, in bytes, seen in the flow.	Yes

flow_pkts_payload.std	The standard deviation of the payload size, in bytes, seen in the flow	Yes
payload_bytes_per_second	The average number of payload bytes transmitted per second. If the duration is 0 then this feature is also set to 0.	Yes
flow_FIN_flag_count	The total number of FIN flags which have been seen in a TCP flow. If the the flow is not a TCP flow this feature is set to 0.	Yes
flow_SYN_flag_count	The total number of SYN flags which have been seen in a TCP flow. If the the flow is not a TCP flow this feature is set to 0.	Yes
flow_RST_flag_count	The total number of RST flags which have been seen in a TCP flow. If the the flow is not a TCP flow this feature is set to 0.	Yes
fwd_PSH_flag_count	The total number of PSH flags which have been seen in the forward direction of a TCP flow. If the the flow is not a TCP flow this feature is set to 0.	Yes
bwd_PSH_flag_count	The total number of PSH flags which have been seen in the backward direction of a TCP flow. If the the flow is not a TCP flow this feature is set to 0.	Yes
flow_ACK_flag_count	The total number of ACK flags which have been seen in a TCP flow. If the the flow is not a TCP flow this feature is set to 0.	Yes
fwd_URG_flag_count	The total number of URG flags which have been seen in the forward direction of a TCP flow. If the the flow is not a TCP flow this feature is set to 0.	Yes

bwd_URG_flag_count	The total number of URG flags which have been seen in the backward direction of a TCP flow. If the the flow is not a TCP flow this feature is set to 0.	Yes
flow_CWR_flag_count	The total number of CWR flags which have been seen in a TCP flow. If the the flow is not a TCP flow this feature is set to 0.	Yes
flow_ECE_flag_count	The total number of ECE flags which have been seen in a TCP flow. If the the flow is not a TCP flow this feature is set to 0.	Yes
fwd_iat.max	The largest inter-arrival time in microseconds bet two consecutive packets in the forward direction.	Yes
fwd_iat.min	The smallest inter-arrival time in microseconds bet two consecutive packets in the forward direction.	Yes
fwd_iat.tot	The inter-arrival time in microseconds bet two consecutive packets in the forward direction.	Yes
fwd_iat.avg	The average inter-arrival time in microseconds bet two consecutive packets in the forward direction.	Yes
fwd_iat.std	The standard deviation of all inter-arrival times in the forward direction in microseconds.	Yes
bwd_iat.max	The largest inter-arrival time in microseconds bet two consecutive packets in the backward direction.	Yes
bwd_iat.min	The smallest inter-arrival time in microseconds bet two consecutive packets in the backward direction.	Yes
bwd_iat.tot	The inter-arrival time in microseconds bet two consecutive packets in the backward direction.	Yes

bwd_iat.avg	The average inter-arrival time in microseconds bet two consecutive packets in the backward direction.	Yes
bwd_iat.std	The standard deviation of all inter-arrival times in the backward direction in microseconds.	Yes
flow_iat.max	The largest inter-arrival time in microseconds bet two consecutive packets in the flow.	Yes
flow_iat.min	The smallest inter-arrival time in microseconds bet two consecutive packets in the flow.	Yes
flow_iat.tot	The inter-arrival time in microseconds bet two consecutive packets in the flow.	No
flow_iat.avg	The average inter-arrival time in microseconds bet two consecutive packets in the flow.	Yes
flow_iat.std	The standard deviation of all inter-arrival times in the flow, in microseconds.	Yes
fwd_subflow_pkts	The average number of packets in the subflows in the forward direction.	Yes
bwd_subflow_pkts	The average number of packets in the subflows in the backward direction.	Yes
fwd_subflow_bytes	The average number of payload bytes in the subflows in the forward direction.	Yes
bwd_subflow_bytes	The average number of payload bytes in the subflows in the backward direction.	Yes
fwd_bulk_bytes	The average number of payload bytes transmitted in a bulk transmission in forward direction.	Yes
bwd_bulk_bytes	The average number of payload bytes transmitted in a bulk transmission in backward direction.	Yes
fwd_bulk_packets	The average number of packets transmitted in a bulk transmission in forward direction.	Yes

bwd_bulk_packets	The average number of packets transmitted in a bulk transmission in backward direction.	Yes
fwd_bulk_rate	The average number of payload bytes transmitted per second during a bulk transmission in forward direction.	Yes
bwd_bulk_rate	The average number of payload bytes transmitted per second during a bulk transmission in backward direction.	Yes
active.max	The longest duration the flow was active in microseconds.	Yes
active.min	The shortest duration the flow was active in microseconds.	Yes
active.tot	The total duration the flow was active in microseconds.	Yes
active.avg	The average duration the flow was active in microseconds.	Yes
active.std	The standard deviation of all active periods in microseconds.	No
idle.max	The longest duration the flow was idle in microseconds.	Yes
idle.min	The shortest duration the flow was idle in microseconds.	Yes
idle.tot	The total duration the flow was idle in microseconds.	Yes
idle.avg	The average duration the flow was idle in microseconds.	Yes
idle.std	The standard deviation of all idle periods in microseconds.	No
fwd_init_window_size	The window size in bytes the first packet in the forward direction has. The windows scale	Yes

	parameter is currently ignored, as this is only set in a SYN packet but we currently look at any packet.	
bwd_init_window_size	The window size in bytes the first packet in the backward direction has. The windows scale parameter is currently ignored, as this is only set in a SYN packet but we currently look at any packet.	Yes
fwd_last_window_size	The window size in bytes the last packet in the forward direction has. The windows scale parameter is currently ignored, as this is only set in a SYN packet but we currently look at any packet.	Yes
bwd_last_window_size	The window size in bytes the last packet in the backward direction has. The windows scale parameter is currently ignored, as this is only set in a SYN packet but we currently look at any packet.	Yes

3.2.4 Characteristics of the Dataset:

Diversity: Almost all most common attacks are carried out such as DDoS, DoS and PortScan etc.
Feature Set: Extracted more than 80 network flow features from the generated network traffic using CICFlowMeter and delivered the network flow dataset as a CSV file. Dataset is both available in CSV and PCAP format.
Protocols: All common protocols are present in the dataset, such as HTTP, HTTPS, FTP, SSH.
Labelled Dataset: Most important characteristic of dataset is this that dataset is fully labelled. They have provided complete information on their website to label pcap data.

3.2.5 Dataset Description:

CICIDS Dataset was generated in five days from Monday to Friday. Dataset consists of five pcap file that are named after the name of the day when they were created.

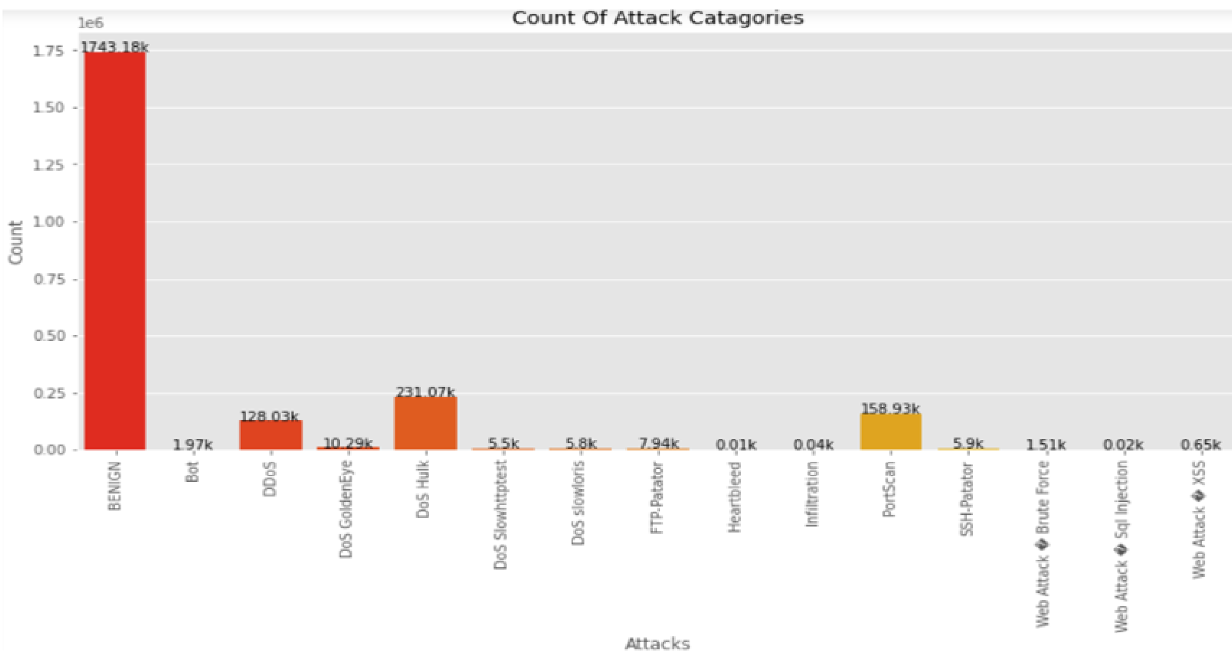
- Monday, Normal Activity, 11.0G (BENIGN Traffic only)
- Tuesday, attacks + Normal Activity, 11G (BruteForce)
- Wednesday, attacks + Normal Activity, 13G (DoS)

- Thursday, attacks + Normal Activity, 7.8G (Web Attacks)
- Friday, attacks + Normal Activity, 8.3G (PortScan + DDoS)

Dataset consists of 84 features (extracted from ZeekFlowmeter) and each record of dataset is a bidirectional flow. Bidirectional flow means that the traffic between two hosts for a duration of time. Based on this traffic 84 features are extracted which are shown in Table 3.1

3.2.6 Dataset Analysis

Some dataset classes are named after the tools to generate dataset. For example, DoS Hulk, DoS Slowloris, DoS slowHTTPtest and DoS GoldenEye are tools to carry out DoS attack. Similarly, FTP-Patator and SSH-Patator are tools to carry out BruteForce attack.



Dataset is highly class imbalanced, and size of dataset is very large. It covers seven types of network attacks. Web attacks are in minority while PortScan, DDoS and DoS-Hulk (type of DoS attack) are in majority. To overcome this problem, we came up with a strategy of class merge and sampling.

We merged the classes of same attack. For example, DoS Hulk, DoS Slowloris, DoS Goldeneye, and DoS slowHTTPtest belongs to DoS attack. We concatenated these four classes and labelled them as DoS. Same as this SSH-BruteForce and FTP-BruteForce are merged and labelled as BruteForce.

Previous Label	Count	After down sampling	New Label	Count
DoS Hulk	163453	20000	DoS	46765
DoS GoldenEye	6932	6932		
DoS slowHTTPtest	15932	15932		
DoS Slowloris	3901	3901		

Previous Label	Count	After down sampling	New Label	Count
FTP-Patator	4032	4032	BruteForce	6972
SSH-Patator	2950	2950		

After merging sub-classes, sampling, and dropping **Infiltration, botnet and web-attacks**, final distribution of dataset is as follows.

Category	Count
BENIGN	150000
DoS	46765
DDoS	25000
PortScan	25000
BruteForce	6972
Total	**253737**

3.2.7 ATTACKS:

NADE is, for the time, trained to detect 4 types of Network-Layer attacks listed as follows:

- Port-Scan
- BruteForce
- DoS
- DDoS

Attacks are further categorized by the techniques used to perform the attack.

3.2.7.1 PORTSCAN:

Attacks are always performed in different phases. The first phase of every attack is the Information Gathering part. The Attacker tries and collects information about the target before actually performing the attacks. This could be any information related to the victim/target. One of the most common parts of this phase is Port Scanning.

When a computer runs a network service, it opens a networking construct called a “port” to receive the connection. Ports are necessary for making multiple network requests or having multiple services available. For example, when you load several web pages at once in a web browser, the program must have some way of determining which tab is loading which web page. This is done by establishing connections to the remote web servers using different ports on your local machine. Network connections are made between two ports – an open port listening on the server and a randomly selected port on your own computer. For example, when you connect to a web page, your computer may open port 49534 to connect to the server’s port 443. There are a total of 65535 available ports on a computer.

Not knowing which ports are open can decrease the attackers’ chance of successfully attacking the target. So, usually, attacks begin with a port scan. Basically in a port scan, the attacker tries to connect to all the ports of the target, and the responses are used to determine if the port is open, closed, or filtered (usually by a firewall).

NADE used nmap port scans for Training and testing of the ML Model. Nmap, also sometimes known as Network-Mapper, is a free and open-source tool for Network and Port scanning. It is also proficient in many other active information gathering techniques. When port scanning with Nmap, there are three basic scan types. These are:

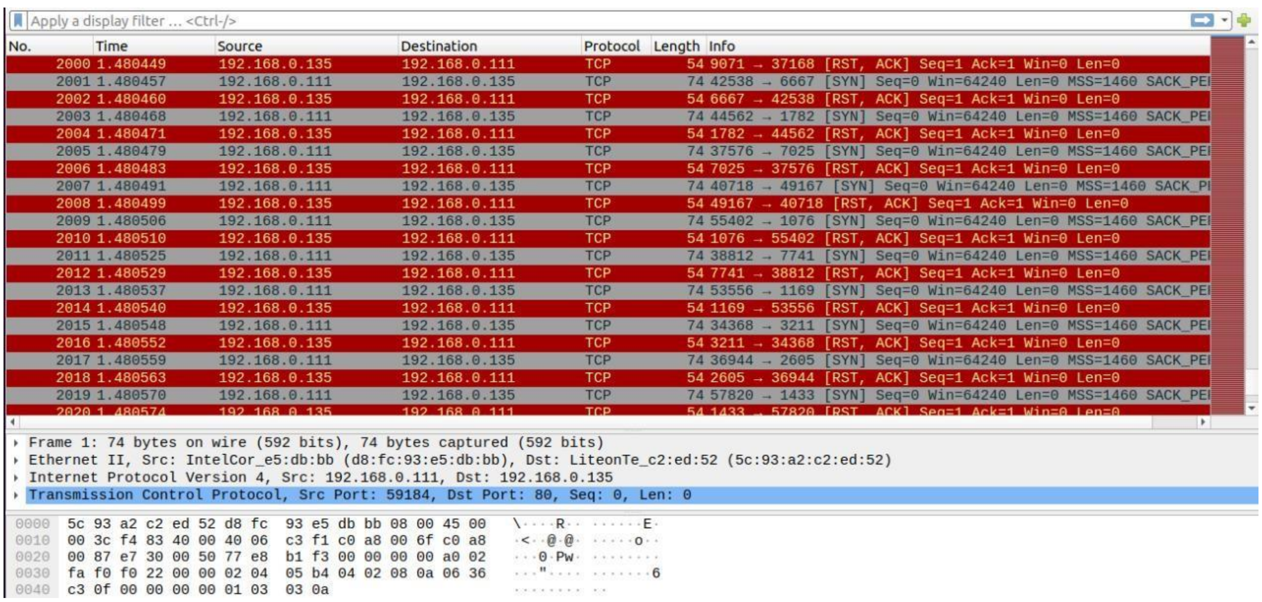
- **TCP Connect Scans (-sT)**

if Nmap sends a TCP request with the *SYN* flag set to a ***closed*** port, the target server will respond with a TCP packet with the *RST* (Reset) flag set. By this response, Nmap can establish that the port is closed. If, however, the request is sent to an *open* port, the target will respond with a TCP packet with the SYN/ACK flags set. Nmap then marks this port as being *open* (and completes the handshake by sending back a TCP packet with ACK set).

What if the port is open, but hidden behind a firewall?

Many firewalls are configured to simply **drop** incoming packets. Nmap sends a TCP SYN request, and receives nothing back. This indicates that the port is being protected by a firewall and thus the port is considered to be *filtered*.

The victim's wireshark looks like this during TCP connect scan attack.



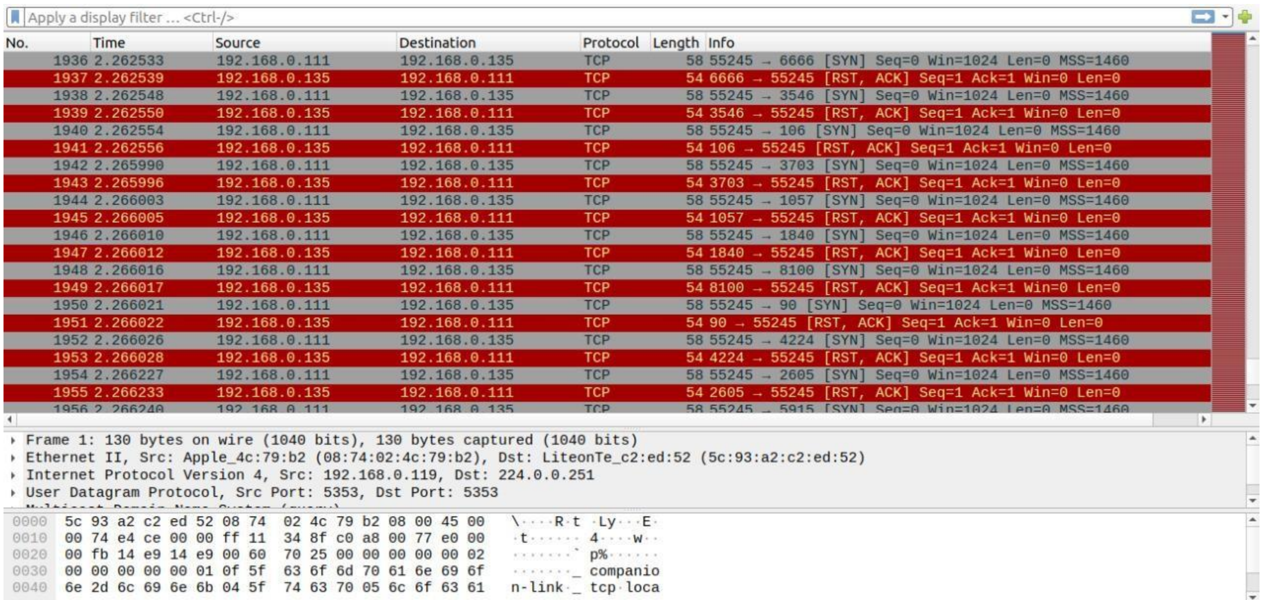
- **SYN "Half-open" Scans (-sS)**

As with TCP scans, SYN scans (-sS) are used to scan the TCP port-range of a target or targets; however, the two scan types work slightly differently. SYN scans are sometimes referred to as "*Half-open*" scans, or "*Stealth*" scans.

Where TCP scans perform a full three-way handshake with the target, SYN scans sends back a RST TCP packet after receiving a SYN/ACK from the server (this prevents the server from repeatedly trying to make the request.

If a port is closed then the server responds with a RST TCP packet. If the port is filtered by a firewall then the TCP SYN packet is either dropped, or spoofed with a TCP reset.

Here is the wireshark analysis of victim Network during the attack



- **UDP Scans (-sU)**

Unlike TCP, UDP connections are *stateless*. This means that, rather than initiating a connection with a back-and-forth "handshake", UDP connections rely on sending packets to a target port and essentially hoping that they make it. This makes UDP superb for connections which rely on speed over quality (e.g. video sharing), but the lack of acknowledgement makes UDP significantly more difficult (and much slower) to scan. The switch for an Nmap UDP scan is (-sU).

When a packet is sent to a *closed* UDP port, the target should respond with an ICMP (ping) packet containing a message that the port is unreachable. This clearly identifies closed ports, which Nmap marks as such and moves on.

When a packet is sent to an open UDP port, there should be no response. When this happens, Nmap refers to the port as being open|filtered

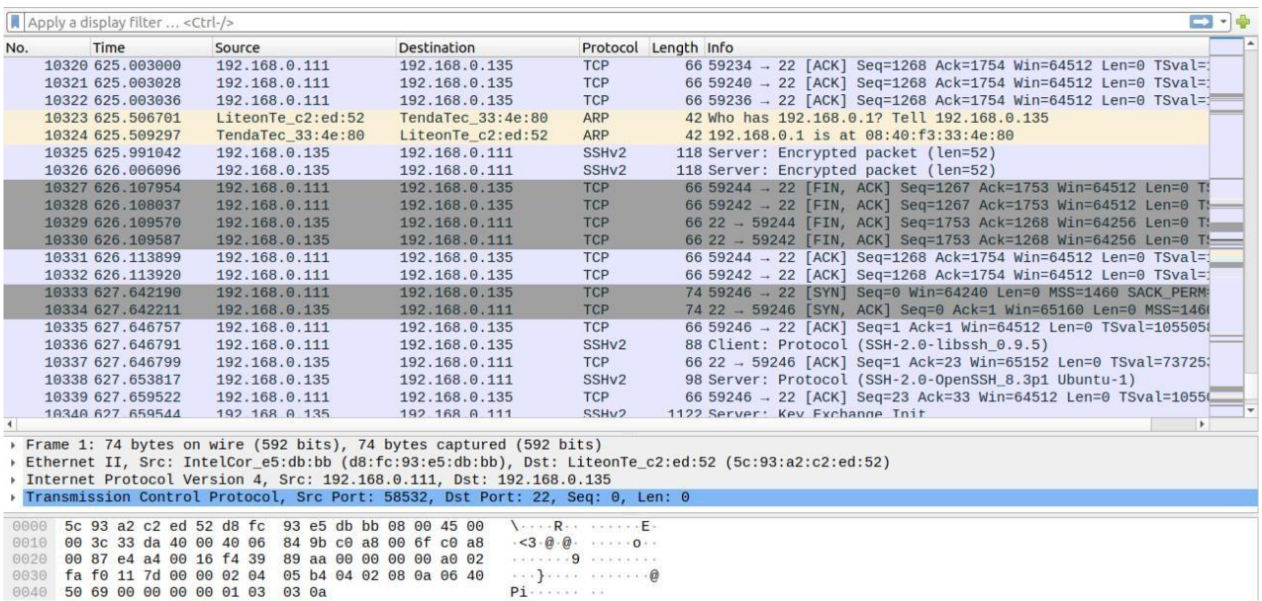
3.2.7.2 BRUTEFORCE

A **brute force attack** uses trial-and-error to guess login info, encryption keys, or find a hidden web page.

NADE covered FTP and SSH bruteforce attacks. These were conducted with Hydra.

Hydra is a parallelized network login cracker built in various operating systems like Kali Linux, Parrot and other major penetration testing environments. **Hydra** works by using different approaches to perform **brute-force** attacks in order to guess the right username and password combination.

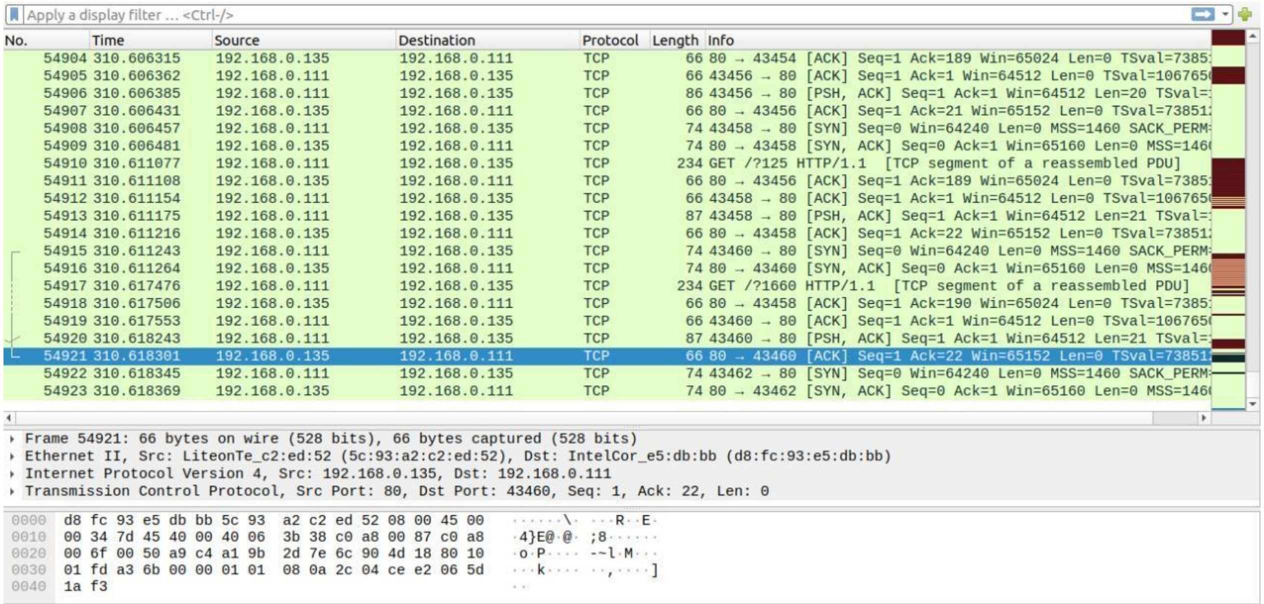
The Wireshark analysis of victim Network during a hydra BruteForce is provided here.



3.2.7.3 DoS

A denial-of-service attack is a cyber-attack in which the perpetrator seeks to make a machine or network resource unavailable to its intended users by temporarily or indefinitely disrupting the services of a host connected to the Internet.

NADE used slowloris and hulk in DoS attacks. The wireshark of victim is given below during slowloris.



3.2.7.4 DDoS

A distributed **denial-of-service** (**DDoS**) **attack** occurs when multiple systems flood the bandwidth or resources of a targeted system, usually one or more web servers.

DDoS attacks are carried out with networks of Internet-connected machines known as a botnet. When a victim's server or network is targeted by the botnet, each bot sends requests to the target's IP address, potentially causing the server or network to become overwhelmed, resulting in a denial-of-service to normal traffic.

3.2.8 MACHINE LEARNING:

3.2.8.1 What is Machine Learning?

Machine Learning is a subfield of Artificial Intelligence which learns from the experience(dataset) and makes prediction on unknown data. Machine Learning is very popular nowadays because of the availability of sophisticated algorithms, large datasets and system resources. Machine Learning algorithms can be classified into two categories.

- **Supervised Learning Algorithms:**

  In supervised learning we provide both data and labels to train the model. Logistic Regression, Decision Tree and Neural Networks are supervised learning algorithms.

- **Un-supervised Learning Algorithms:**

In unsupervised learning model is trained on unlabeled data. K-mean clustering is one of the mostly used unsupervised learning algorithms.
As our project is intended to find out anomaly and classify it so we have to use supervised Learning algorithms.

3.2.8.2 Machine Learning in Cybersecurity:

In conventional or rule-based IDS we have to identify patterns of attacks and define rules manually to detect abnormal behavior in the network. But in Machine Learning-based IDS rules or patterns are made automatically. In order to apply Machine learning we need huge data. As networks are growing rapidly and a large amount of data is being generated every day, so we have enough data to train our machine learning model and extract rules out of that. Once a machine learning model is trained, we can deploy it in a network where it can make real time prediction/detection.

3.2.8.3 Our Implementation of Machine Learning:

We have implemented supervised machine learning in our project. CIC dataset is a labeled dataset that means we can train a supervised learning model for attack detection. We have five classes in our final dataset which means our trained model should classify incoming traffic among these five classes.

As our machine learning model will make predictions in real time, we need a model which takes minimum time to predict and has maximum accuracy. We experimented many machines learning models and techniques. We found out that Decision Tree Classifier suits best in our case as it predicted 25000 records in 0.008 seconds with 99 percent accuracy. We tested more classifiers than mentioned here. These classifiers showed best results.

3.2.8.4 Machine Learning Classifiers:

Machine learning classifiers used in this project are briefly described here,

Logistic Regression: is an iterative machine learning algorithm which consists of a perceptron and have loss function. It tries to optimize the loss (or error) [8].

Quadratic Discriminant Analysis (QDA) uses bayes theorem and covariance matrices for each class to classify new observations [9, p. 149].

Multi-Layer Perceptron (MLP) also called Deep Learning, are inspired by the behavior of neurons in brains. Logically they consist of interconnected nodes in layers that performs calculations to make classifications. Layers and nodes in the network are hyperparameters [10].

Decision Tree (DT) classifies data by traversing through a tree structure, asking relevant questions about the features of the data, when the traversing reaches a leaf node, the data point is classified according to the class in the leaf node [9, p. 303]. ID3 is a popular version of DT.

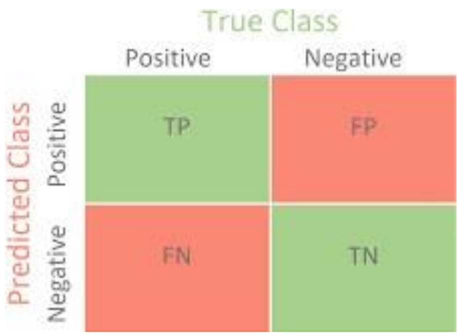
Random Forest (RF) aggregates and produces a mean of the result of several Decision Trees trained from different random subsets of the training data [9, p. 319].

3.2.9 RESULTS:

- **Confusion Matrix**

A confusion matrix is a way of describing the performance of a classification system. For example, if the number of observations in each class is imbalanced or the dataset comprises more than two classes, classification accuracy alone may be misleading. Calculating a confusion matrix can assist us in determining what the categorization model gets right and where it goes wrong.

True Positives: Positive records correctly classified by the model as positive.
True Negatives: Negative records correctly classified by the model as negative.
False Positives: Negative records incorrectly classified by the model as positives.
False Negatives: Positive records incorrectly classified by the model as negatives.



A good IDS must have lower False Positives and False Negatives.

- False Positive means your system is blocking benign traffic.

False Positive Rate (FPR) = FP/FP+TN

- False Negative means your system is allowing malicious traffic to enter the network.

False Negative Rate (FNR) = FN/FN+TP

If the system has a low detection rate, it gives a false sense of security, and if it has a high false alarm rate, the security managers time is wasted analyzing false alarms. The point here is that False positives generates false alarms and hence availability of the server is compromised. Because when benign traffic which is coming from legitimate users is halted (after detected as attack by IDS), It is more dangerous than allowing malicious traffic to enter the network. Because we cannot deny access to users at any cost. So, the focus here is to reduce false positives (false alarms).

- **Train, Validation and Test Sets:**

Training is the most important part in Machine Learning cycle. In training, dataset and labels are fed into model and model learn the rules in the dataset. Dataset is split into training, testing and validation set. Model is trained on the training data and its performance is measured on validation and test set. Our final dataset consists of **253737** records. We split the dataset with 75 / 15 / 10 percent ratio in training, validation, and testing set, respectively.

Validation set is used for a special purpose in machine learning. Almost every ML model is prone to overfitting. Overfitting is a condition in which model shows high accuracy on training data and very poor accuracy on test data. To detect overfitting, we use k-fold cross-validation.

Validation data is split into ‘k’ number of subsets randomly and each subset is passed to trained model then its accuracy is calculated. After we have calculated all k subsets, we average their accuracies and show it as validation accuracy. If validation accuracy is near to training accuracy, our model is not overfitting on training data but if validation accuracy is far from training accuracy, model is overfitting on training data.

	Split Ratio	Count
Training set	0.75	190302
Validation set	0.15	38061
Testing set	0.10	25374

3.2.10 Classifiers Performance:

```
# Models

tr_pre, te_pre, val_pre, tr_acc, te_acc, val_acc  = 0,0,0,0,0,0
classes = ['BENIGN','DoS','PortScan','DDoS','BruteForce']

names = [
         "Logistic Regression",
        "Decision Tree",
        "Random Forest",
         'SGD',
        "AdaBoost",
         'MLP',
         "QDA"
        ]

classifiers = [
    LogisticRegression(),
    DecisionTreeClassifier(),
    RandomForestClassifier(),
    SGDClassifier(),
    AdaBoostClassifier(),
    MLPClassifier(),
    QuadraticDiscriminantAnalysis()
    ]

models = {}

for n,c in zip(names,classifiers):
    models[n] = [c, 'dt', tr_pre, te_pre, val_pre, tr_acc, te_acc, val_acc]

for model in models:

    models[model][1] = models[model][0]
    models[model][1].fit(x_train, y_train_o)
```

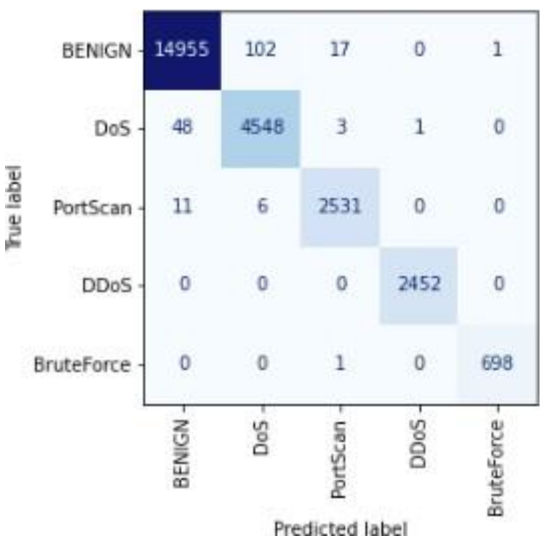
```
    models[model][2] = models[model][1].predict(x_train)
    s=time.time()
    models[model][3] = models[model][1].predict(x_test)
    e=time.time()
    models[model][7] = round(cross_val_score(models[model][1], x_val, y_val_o, cv=5).mean(), 3)

    models[model][5] = round(accuracy_score(y_train_o, models[model][2]), 3)
    models[model][6] = round(accuracy_score(y_test_o, models[model][3]), 3)

    print('Model:',model)
    print('Train Acc:',models[model][5])
    print('Val Acc:',models[model][7])
    print('Test Acc:',models[model][6])
    print('Test Prediction Time:',round(e-s,3))
    plot = plot_confusion_matrix(models[model][1], x_test, y_test_o,
                     display_labels=classes, colorbar=False,
                    labels=classes, cmap=plt.cm.Blues,
                              xticks_rotation='vertical')
    plt.show()
    print('\n')
```

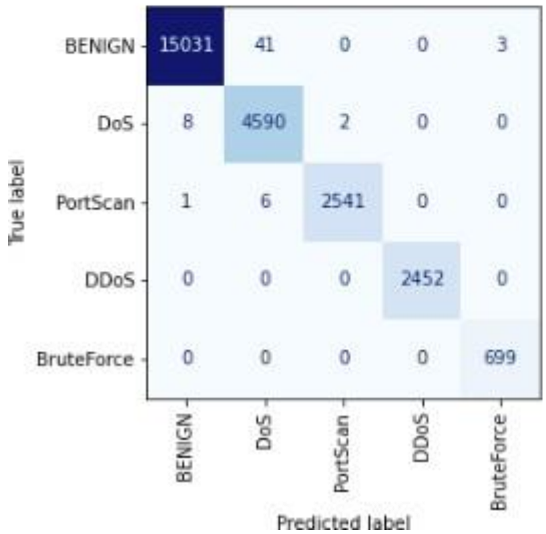
- **Logistic Regression Classifier:**

```
Model: Logistic Regression
Train Acc: 0.993
Val Acc: 0.993
Test Acc: 0.993
Test Prediction Time: 0.01
```



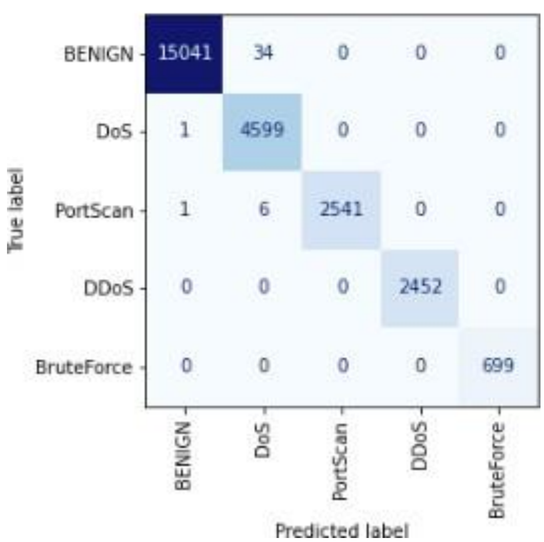
- **Decision Tree Classifier:**

```
Model: Decision Tree
Train Acc: 0.999
Val Acc: 0.996
Test Acc: 0.998
Test Prediction Time: 0.005
```



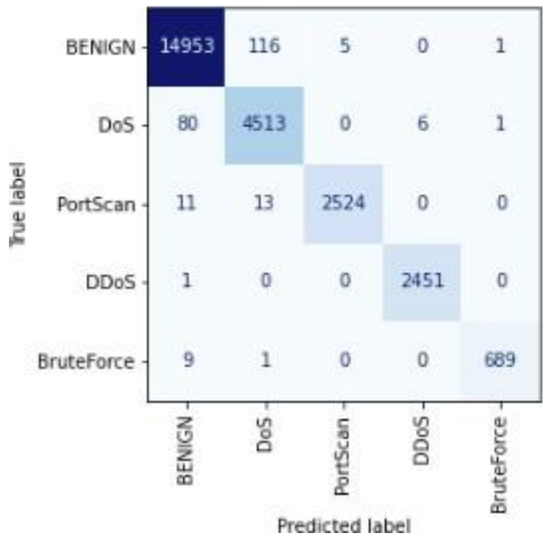
- **Random Forest Classifier:**

```
Model: Random Forest
Train Acc: 0.999
Val Acc: 0.998
Test Acc: 0.998
Test Prediction Time: 0.275
```



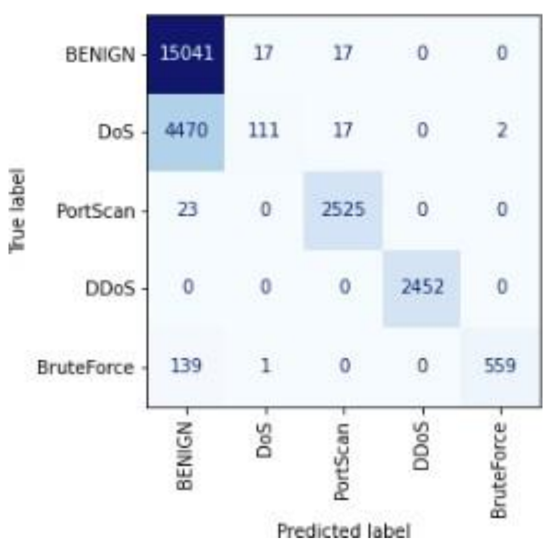
- **SGD Classifier:**

```
Model: SGD
Train Acc: 0.99
Val Acc: 0.99
Test Acc: 0.99
Test Prediction Time: 0.008
```



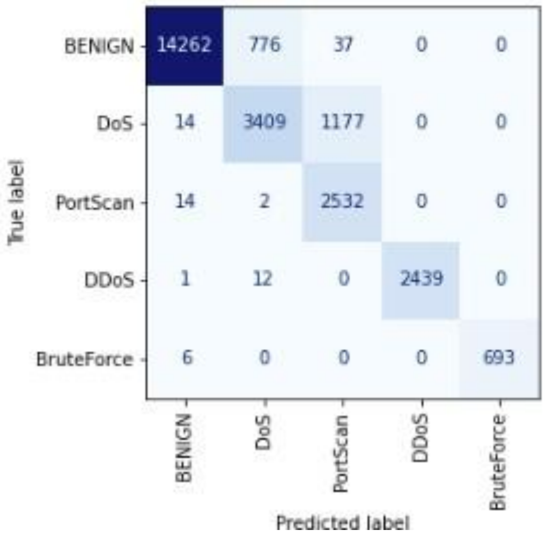
- **AdaBoost Classifier:**

```
Model: AdaBoost
Train Acc: 0.811
Val Acc: 0.765
Test Acc: 0.815
Test Prediction Time: 0.249
```



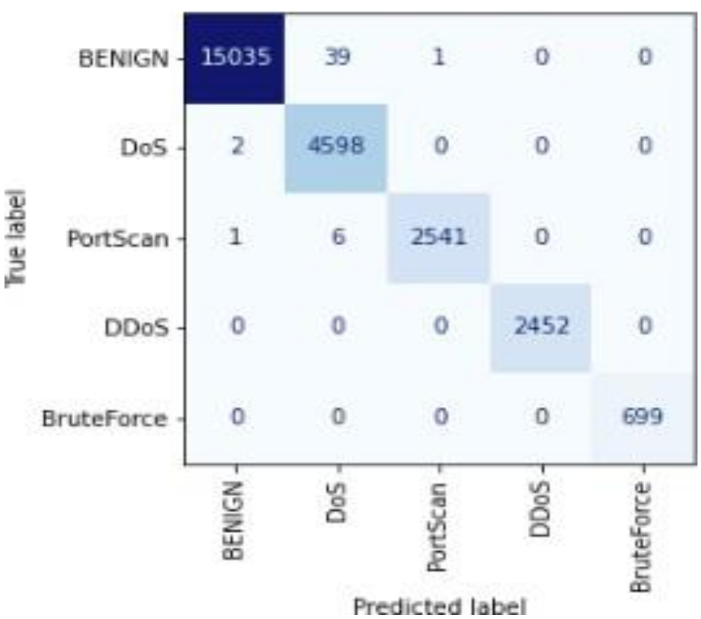
- **QDA Classifier:**

```
Model: QDA
Train Acc: 0.921
Val Acc: 0.927
Test Acc: 0.92
Test Prediction Time: 0.083
```



- **MLP Classifier:**

```
Model: MLP
Train Acc: 0.998
Val Acc: 0.998
Test Acc: 0.998
Test Prediction Time: 0.057
```



Results:

Model	Train Acc	Validation Acc	Test Acc	Test Records	Test Time
Logistic Regression	0.993	0.993	0.993	25000	0.01
Decision Tree	0.999	0.999	0.998	25000	0.005
Random Forest	0.999	0.998	0.998	25000	0.275
SGD Classifier	0.999	0.998	0.998	25000	0.008
AdaBoost	0.811	0.765	0.815	25000	0.249
Multi-Layer Perceptron	0.998	0.998	0.998	25000	0.057
Quadratic Discriminant Analysis	0.921	0.927	0.92	25000	0.083

Here is classification report of decision tree trained model,

Classes	Precision	Recall	F1-Score	Support
BENIGN	1.00	1.00	1.00	15075
BruteForce	0.99	1.00	1.00	699
DDoS	1.00	1.00	1.00	2452
DoS	0.99	1.00	0.99	4600
PortScan	1.00	1.00	1.00	2548

Real Time Results:

We carried out BruteForce, DoS, and Port Scanning attacks in real-time environment and found following results. We set up two machines one is attacker (Kali Linux) and other is victim (Ubuntu). Although, network configuration is not as same as the configuration of the network in which training data was generated. But still BruteForce and PortScan has shown good results. One thing to be noted is this that False Alarm rate is very low which is a big achievement because IDS most of the time misclassify benign traffic as attack traffic [11].

"***In fact, it has been estimated that up to 99% of alerts reported by IDSs are not related to security issues (towards reducing false positives-page 1) [11]***"

Classes	Accuracy	False Alarms (False Positive Rate)	Detection
BENIGN	0.97	0.03	-
BruteForce	0.70	-	0.85
DDoS	-	-	-
DoS	0.40	-	0.25
PortScan	0.98	-	0.95

3.3 Phase 3:

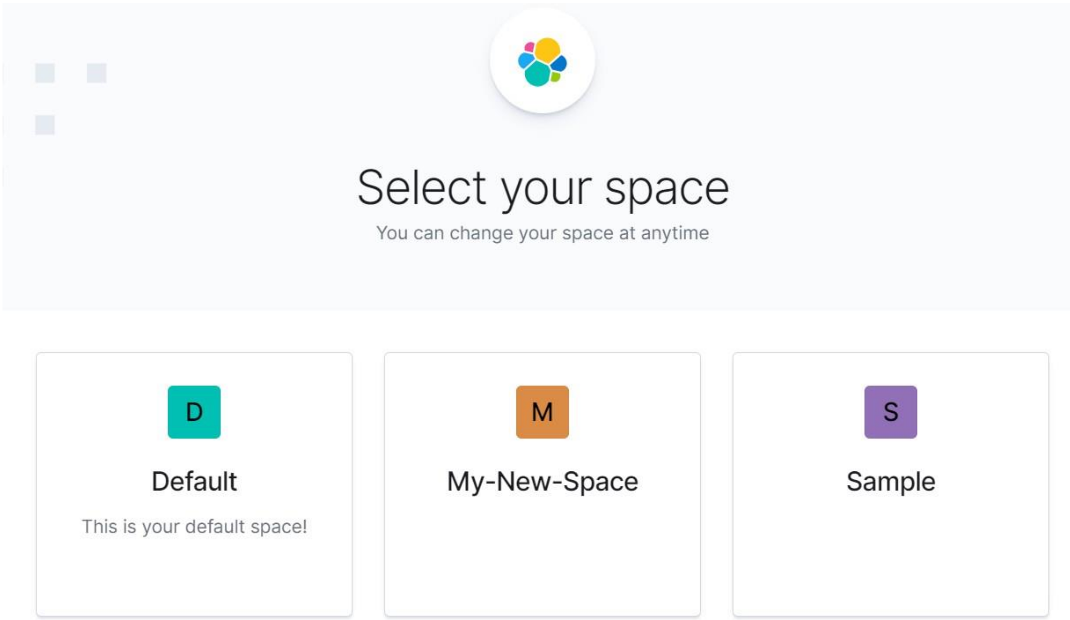
3.3.1 Elastic Search:

Elasticsearch is a part of ELK stack which is an open-source tool kit. Elasticsearch is a distributed, free, and open search and analytics engine for all types of data, including textual, numerical, geospatial, structured, and unstructured. The speed and scalability of Elasticsearch and its ability to index many types of content mean that it can be used for several searches.

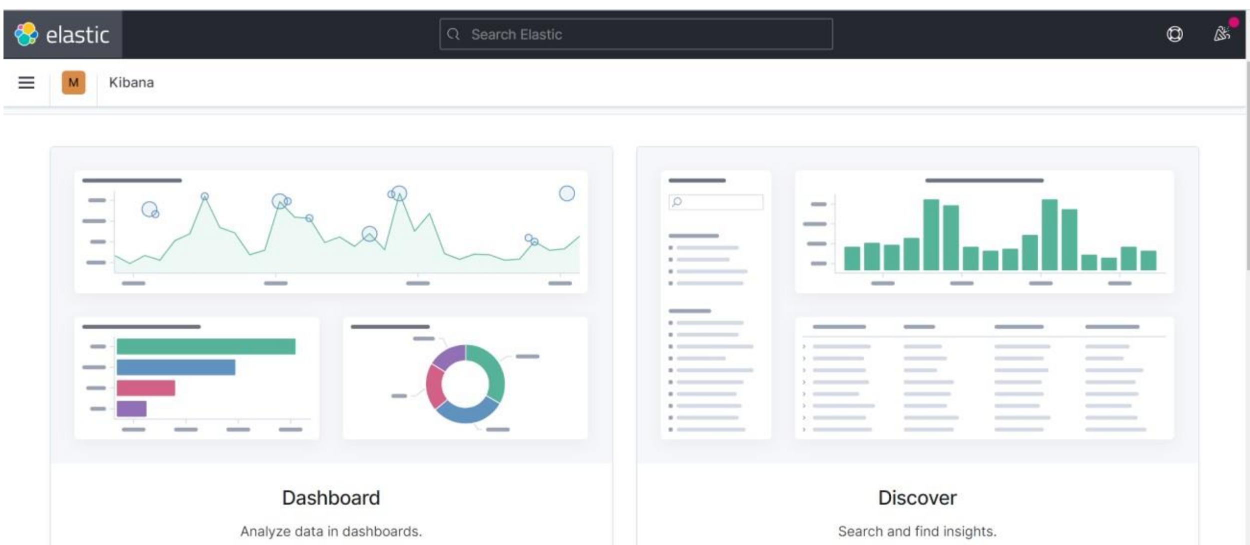
In last phase of our project, the predicted results from phase 2 are sent to Elasticsearch for indexing. NADE creates an index in elastic search, which can be mapped to different columns as per requirement. Once the prediction is sent, network traffic is captured again for 'duration' seconds and this loop continues until it is interrupted.

3.3.2 Kibana:

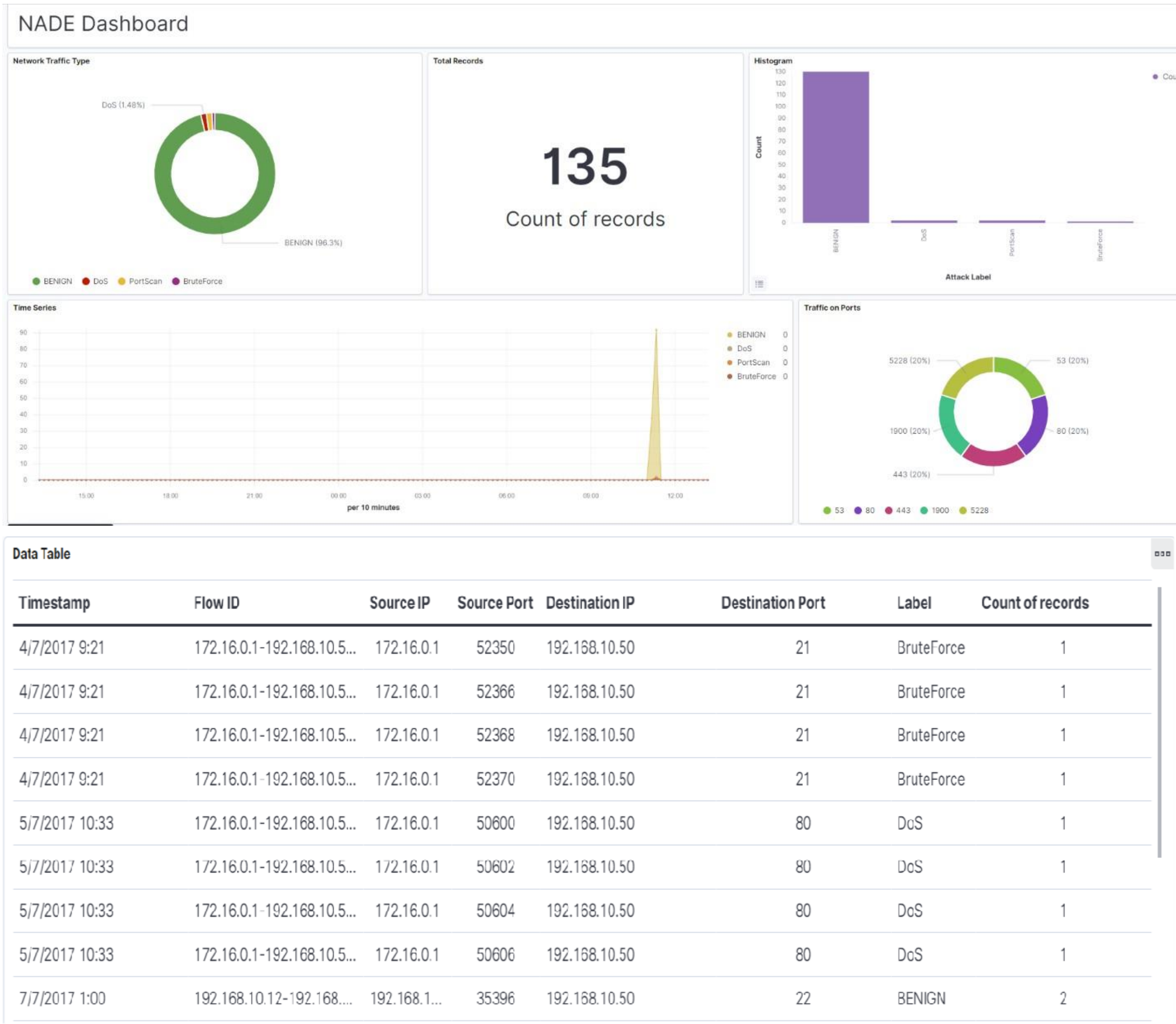
Kibana is a free and open frontend application that sits on top of the Elastic Stack, providing search and data visualization capabilities for data indexed in Elasticsearch. Kibana also acts as the user interface for monitoring and managing data. We can create different spaces for different users.



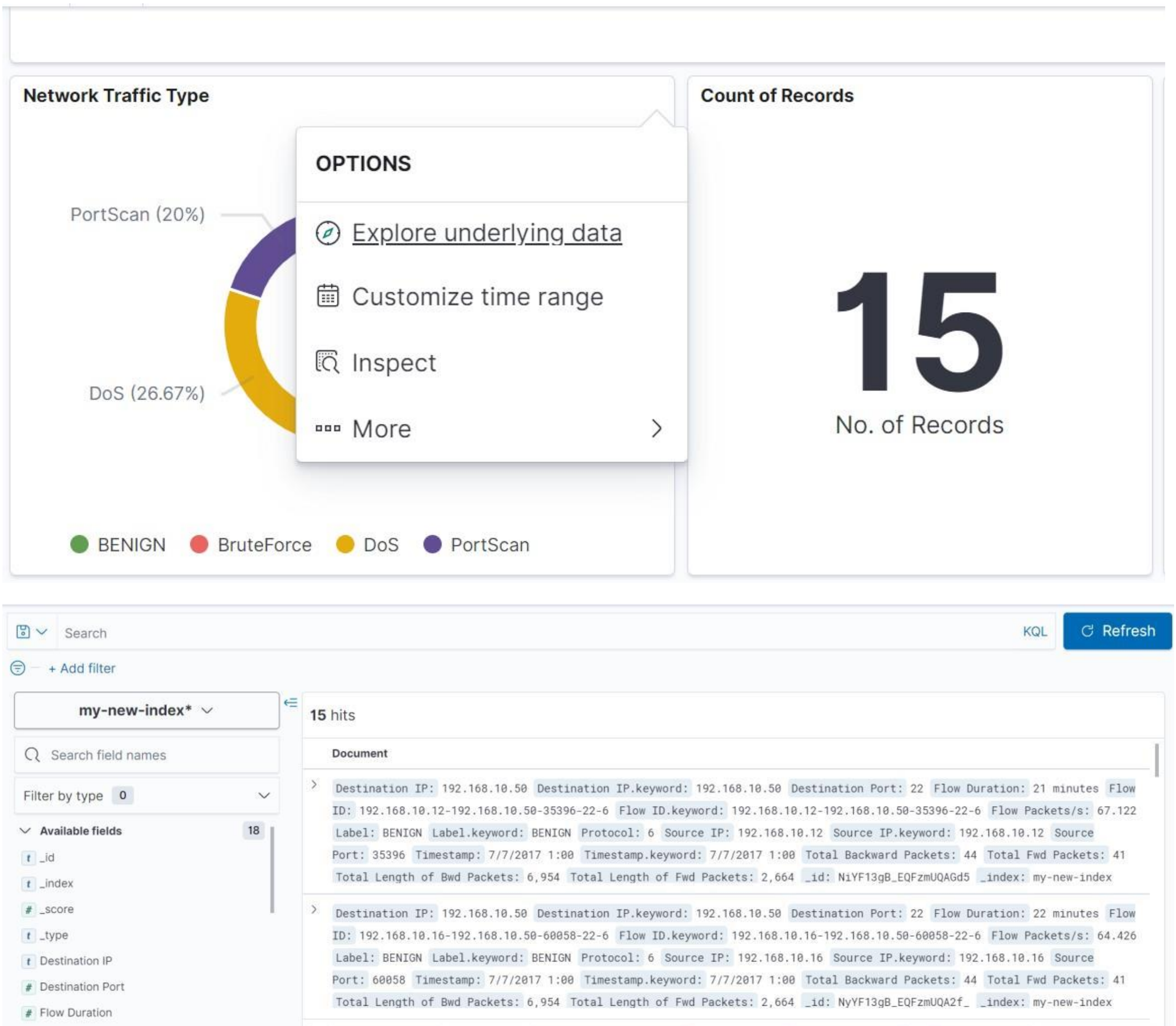
We can create and discover insights of our data.



Kibana is also a part of ELK stack which is a graphical visualization dashboard. We can create multiple visualizations based on our data and requirement. It can be configured to refresh itself after a specific interval of time. Data from elastic search is ingested into Kibana which visualize data in different charts, graphs, histogram etc.



We can explore underlying data of each visualization.



Cost of Product:

EXPENSES	COSTS
Sales per Unit	80,000 Rs.
Customer Service (Installation + Training + Customization)	50,000 Rs.
Development	30,000 Rs.

CHAPTER 4 : SOFTWARE REQUIREMENT SPECIFICATION

4.1 Introduction

The introduction of the Software Requirements Specification (SRS) section provides an overview of the entire SRS with purpose, scope, definitions, acronyms, abbreviations, references, and overview of the SRS. The aim of this document is to present detailed description of the project NADE (Next-Gen Anomaly Detection) which uses a machine learning model to detect advance network security attacks in real-time. The detailed requirements of the NADE are provided in this document.

4.1.1 Purpose

This document covers the software requirement specifications for project “NADE”. The idea of the project is to develop an indigenous network security solution that will provide deeper insight about attack tactics, more clear realization, and visualization of threat landscape. This section is meant to outline the features and requirements of NADE, to serve as a guide to the developer on one hand and a software validation document for the prospective client on the other.

4.2 System Overview

4.2.1 Product Perspective

The size and complexity of today's enterprises is growing exponentially, along with the number of IT personnel to support them. This makes information sharing and collaboration difficult when problems occur. NADE allows security teams to keep on top of security alerts in real-time. By gathering events from all the sources across the network, a NADE can reconstruct the series of events to determine the nature of attack. The main goal of NADE is to improve security, visibility and actionability along with an in-depth analysis of incoming and outgoing traffic. NADE will use custom Zeek scripts to extract useful features from network traffic that will include both attack and benign network data. Then NADE will use Machine Learning driven techniques to detect advanced threats which includes scanning, DoS attacks and other Network layer attacks. Our solution, the Network Generation Anomaly Detection Engine (NADE) will provide a platform where all logs are gathered, and unusual behavior is detected, and attacks are visualized which would add value to the overall security posture of the organization where it is applied.

4.2.2 User Classes and Characteristics

The following section describes the types of users of Network Anomaly Detection Engine (NADE). There are explanations of the user followed by the interactions the user(s) shall be able to make with the software.

4.2.2.1 Network Administrator

People who implement and enforce the company's security program. They are non-hostile and appropriately trained to use, configure, and maintain the software and follow all guidance.

Network Anomaly Detection Engine (NADE) gives network security professionals an in-depth analysis of incoming and outgoing traffic and detect any abnormal behavior. Network Anomaly Detection Engine (NADE) will log the actions that users take on the network, create a baseline to train the ML-driven model which will eventually facilitate the detection, and ultimate halting of advanced network attacks

4.2.3 Operating Environment

The essential physical components for the proper operation of NADE in the evaluated configuration are:

4.2.3.1 Software

- IDE: Python IDE (python 3)
- OS: Linux, Windows
- Dashboard: Kibana
- Elastic Search for logging
- Machine Learning libraries: Sklearn, pandas, numpy
- Zeek

4.2.3.2 Hardware

- Workstation (for training)
- Standard Desktop Client

4.2.4 Design and Implementation Constraints

Language requirements: Software must be in English language

4.2.5 User Documentation

Following are the guides for the user of NADE:

- User Manual
- Online Documentation for users

Documentation for developers and technicians working on the projects include:

- Project Synopsis
- SRS Document
- UML Diagrams/Documents

4.2.6 Assumptions and Dependencies

- NADE will deal with only network layer attacks.
- There are one or more competent individuals assigned to manage NADE.
- Authorized administrators who manage NADE are non-hostile and are appropriately trained to use, configure, and maintain, the TOE, and follow all guidance.
- There is the possibility of detection of false positives and false negatives.
- It is assumed that the IT environment will provide a secure line of communication between distributed portions of the NADE and between the NADE and remote administrators.

4.3 External Interface Requirements

4.3.1 Hardware Interfaces

- Computers/Laptops with Internet Connections

4.3.2 Software Interfaces

- Operating System: Windows / Linux
- Frontend Dashboard: Kibana
- Deployment of Trained Model on Server (if required)

4.3.3 Communications Interfaces

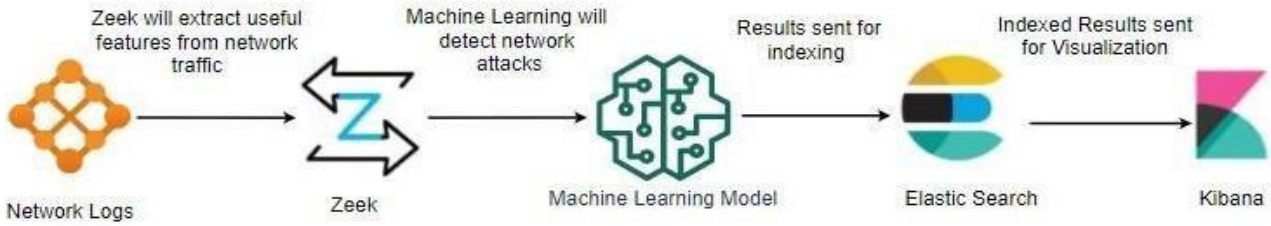
Wi-Fi/Ethernet will be used by the client to connect to the server on which the trained model is present.

4.4 System Features

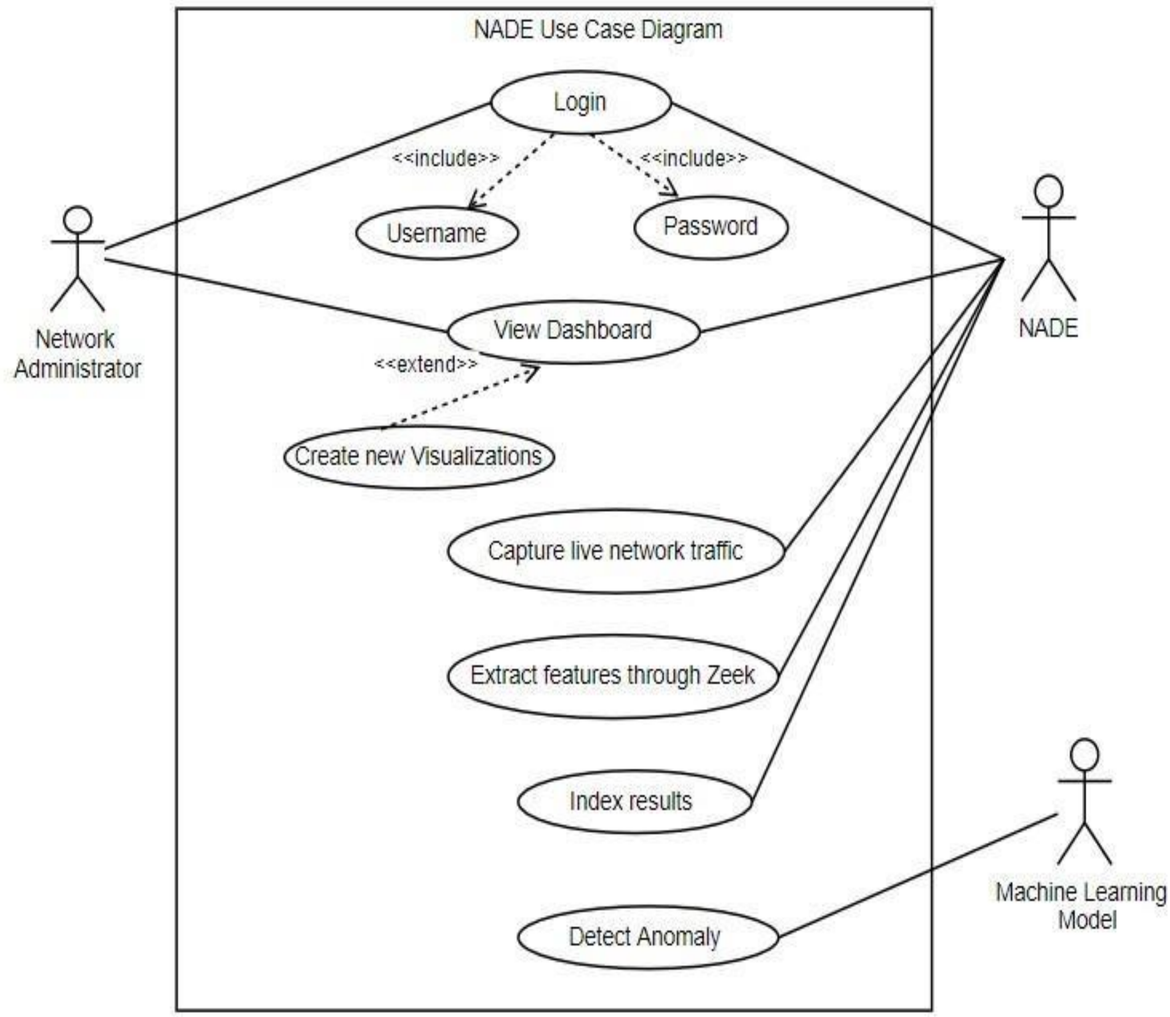
This section illustrates organizing the functional requirements for the NADE:

- Authentication
- Capture Live Network Traffic
- Feature Extraction through Zeek
- Anomaly Detection using Machine Learning
- Data Indexing using Elastic Search
- Visualization through Kibana

Following is the component Diagram:



4.5 Overall Use Case Diagram



4.5.1 Authentication

4.5.1.1 Description

The admin has to login on ELK stack (Kibana dashboard) by entering username and password.

4.5.1.2 Stimulus/Response Sequence

Normal Path: Admin is logged in successfully.
Preconditions • The user enters the username and password.
Interactions • Entered information is checked against registered users.
Post conditions • User is successfully directed to dashboard.
Categorization • **Criticality:** High • **Probability of Defects:** Low • **Risk:** High

Exceptional Path: Entered Username and Password are incorrect.
Preconditions • User entered the wrong username and password.
Interactions • The entered parameters are checked against registered users.
Post conditions • An error notification is generated that you entered an incorrect username or password.
Categorization • **Criticality:** High • **Probability of Defects:** Low • **Risk:** High

4.5.1.3 Functional Requirements

1. The system shall be able to login user.
2. The system shall be able to authenticate user.
3. In case the information entered is incorrect, then it shall generate an error notification.

4.5.2 Capture Live Network Traffic

This feature enables the system to capture network traffic in real time and generate logs. These logs will be fed into the system for further processing.

4.5.2.1 Stimulus/Response Sequences

Normal Path: Logs Retrieval Successfully
Preconditions • There is some sort of traffic on Network.
Interactions • Network traffic is captured in real time and logs are generated
Post conditions • Logs retrieved successfully.
Categorization • **Criticality:** High • **Probability of Defects:** Low • **Risk:** High

Exceptional Path: Logs not Generated
Preconditions • There is an error in Network.
Interactions • Network traffic is not captured, and logs are not generated.
Post conditions • Logs generation failed.
Categorization • **Criticality:** Very High • **Probability of Defects:** Low • **Risk:** High

4.5.2.2 Functional Requirement

1. The system shall be able to capture network traffic.
2. If successful, proceed to next step (feature extraction).

4.5.3 Feature Extraction through Zeek

4.5.3.1 Description

Useful features from the logs obtained from previous stage that will include both attack and benign network data will be extracted through Zeek scripts.

4.5.3.2 Stimulus/Response Sequence

Normal Path: Features are extracted using Zeek
Preconditions • The logs are successfully retrieved.
Interactions • Useful features are extracted from network traffic using Zeek scripts.
Post conditions • Features are extracted successfully.
Categorization • **Criticality:** High • **Probability of Defects:** Medium • **Risk:** High

4.5.3.3 Functional Requirements

1. Features are extracted based on which we will detect attacks.
2. The Zeek Script shall be able to extract useful features.

4.5.4 Anomaly Detection using Machine Learning

4.5.4.1 Description

Based on the outputs from the previous stage, the trained machine learning model shall give a final classification whether the network is under attack or not.

4.5.4.2 Sequence/Response Sequences

Normal Path: Attacks are detected successfully
Preconditions • Standardized features are extracted from previous stage and analyzed.
Interaction • Standardized features from previous stage are fed into the trained Machine Learning model. The machine learning model detect any anomaly in the Network.
Post conditions • The user is notified that there is an attack.
Categorization • **Criticality:** High • **Probability of Defects:** Medium • **Risk:** High

4.5.4.3 Functional Requirements

1. The model shall be able to detect any malicious attacks and anomaly in the Network.
2. The model shall be able to discard the false positives.
3. The model shall be able to detect the type of true positives.

4.5.5 Data Indexing

4.5.5.1 Description

Results obtained from previous stage will be indexed for efficient access and visualization.

4.5.5.2 Stimulus/Response Sequence

Normal Path: Data is indexed using Elastic Search.
Preconditions • Data is ingested into elastic search and rules are defined to control dynamic mapping and explicitly define mappings to take full control of how fields are stored and indexed.
Interactions • Elastic search indexes the data and make it searchable based on defined mappings.
Post conditions • Data is indexed successfully.
Categorization • **Criticality:** High • **Probability of Defects:** Medium • **Risk:** High

Exceptional Path: Data is not indexed using Elastic Search
Preconditions • Elastic search is not running,
Interactions • The data ingestion and indexing will fail.
Post conditions • No logs will be shown on Kibana Dashboard and error message is displayed.
Categorization • **Criticality:** High • **Probability of Defects:** Medium • **Risk:** High

4.5.5.3 Functional Requirements

1. Data can be ingested into Elastic Search.

2. Data is indexed on the bases on defined mappings.
3. In case the elastic search is not running, an error message is displayed.

4.5.6 Data Visualization

4.5.6.1 Description

The logs obtained from previous stage will be visualized in Kibana dashboard.

4.5.6.2 Stimulus/Response Sequence

Normal Path: Data is visualized successfully.
Preconditions • The data is indexed.
Interactions • Logs will be visualized using different graphs and charts.
Post conditions • Dashboard is ready for admin.
Categorization • **Criticality:** High • **Probability of Defects:** Medium • **Risk:** High

Exceptional Path: Data is not visualized successfully.
Preconditions • Elastic search or Kibana is not running.
Interactions • Without elastic search, Kibana won't run and data will not be displayed .
Post conditions • An error message is shown to admin.
Categorization

- **Criticality:** High
- **Probability of Defects:** Medium
- **Risk:** High

4.5.6.3 Functional Requirements

1. Data can be visualized in different ways using graphs, pie chart, histogram etc.
2. In case the Kibana is not running, then the system shall generate an error message.

4.6 Other Non-Functional Requirements

4.6.1 Performance Requirements

As performance is the critical component in security solutions so NextGen Anomaly Detection Engine (NADE) is designed to reduce the delay in transmission. NADE will operate in real time environment.

4.6.2 Safety Requirements

An authorized user of the internal network must not be able to transmit sensitive data outside of the internal network, exposing it to unauthorized viewers.

An authorized user of the internal network must not temper with the records of NADE as Persons may not be held accountable for their changes to the data because their actions are not recorded.

4.6.3 Security Requirements

The System will ensure that only authorized administrators are granted access to the security functions, configurations, and associated data.

4.6.4 Software Quality Attributes

4.6.4.1 Availability

The endpoints should be up and running 24/7.

4.6.4.2 Correctness:

The logs generated must always have correct time stamp.

4.6.4.3 Accuracy:

NADE will provide more accurate results by using best possible Machine Learning Model.

4.6.4.4 Adaptability:

Currently the server runs on windows OS, but it must adapt to Linux/Unix.

4.6.5 Business Rules

- The system is available for linux and windows only.
- NADE solution is only capable to tackle network attacks. (Port sanning, Dos attack and other network layer attacks)
- NADE solution is suitable for small to medium networks.

CHAPTER 5 : DESIGN AND DEVELOPMENT

5.1 Introduction

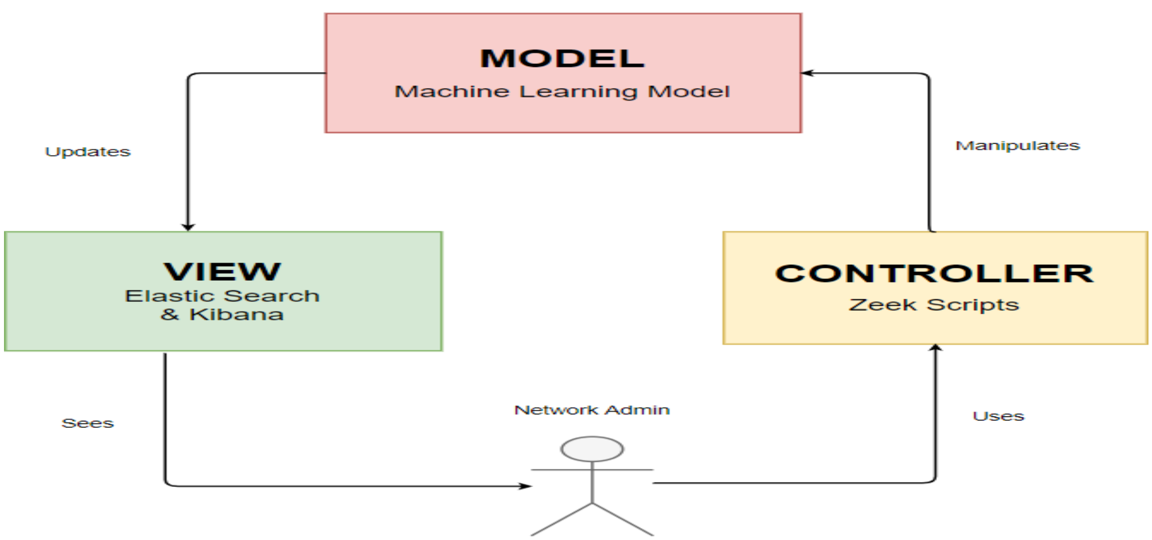
5.1.1 Purpose

This software design section contains the complete design description of the project *"Network Anomaly Detection Engine (NADE)"*. The purpose of this document is to understand each component and module of the project. It will provide information about the relationship between each module and how they are interconnected. The document is intended to inform stakeholders the details of the design and the design process. It is meant to outline the features, structure, and architecture of a "NADE", to serve as a guide to developers and the intended audience. The intended audiences for the NADE include project supervisor, group members, project evaluation team and other concerned persons. It also shows how the use cases detailed in the SRS will be implemented in the system using this design.

5.2 System architecture

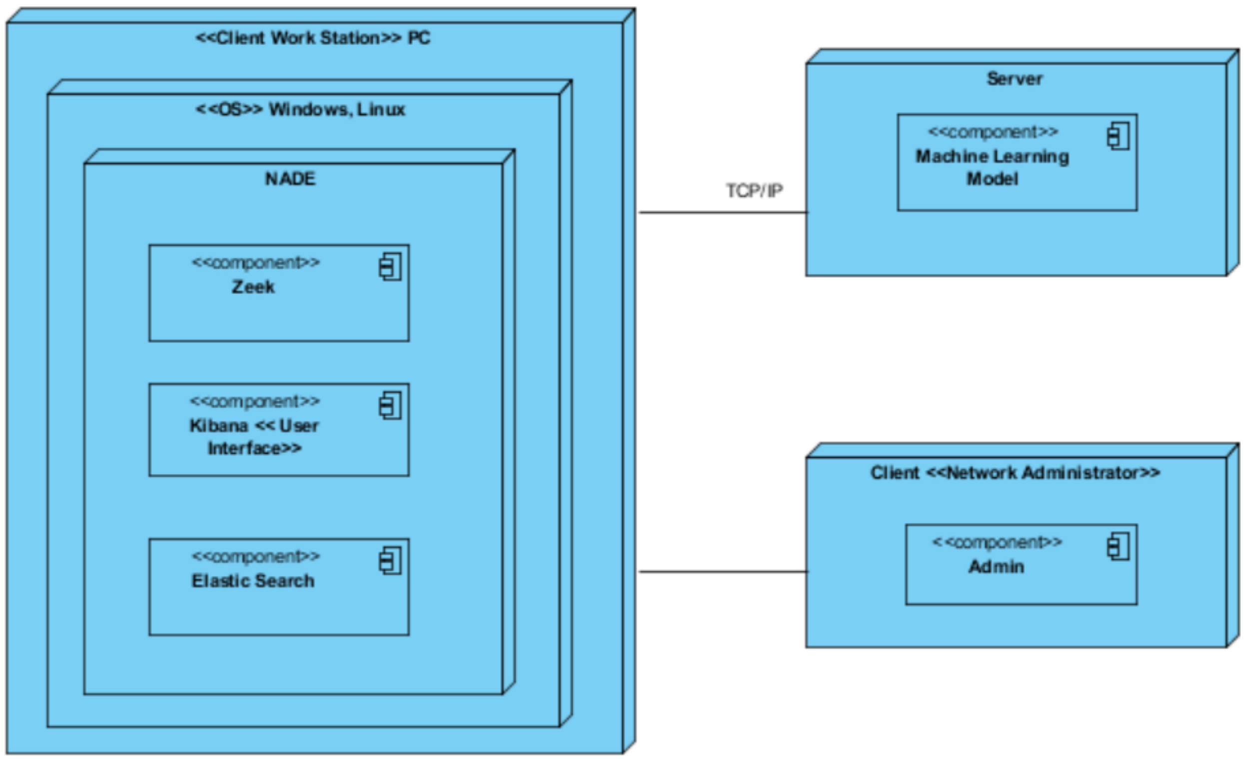
5.2.1 Architectural Design

The diagram below provides an illustration of the System architecture along with the various components used.

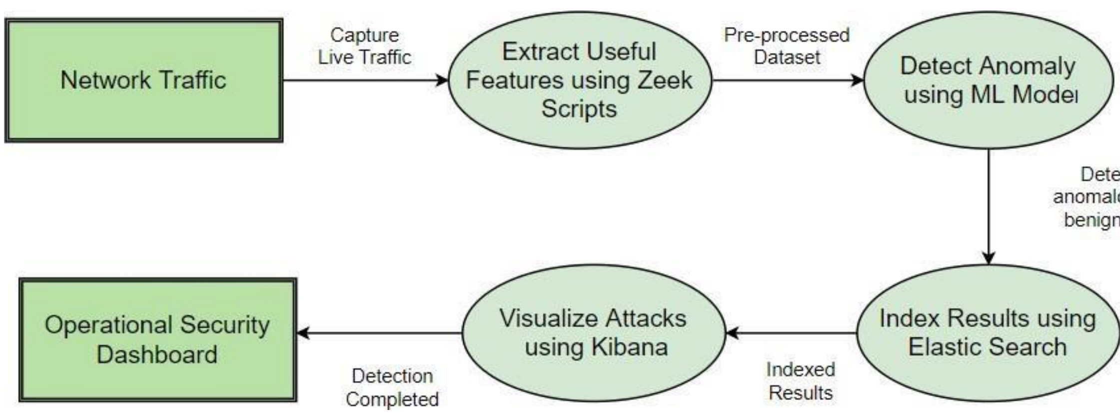


5.2.2 Decomposition Description

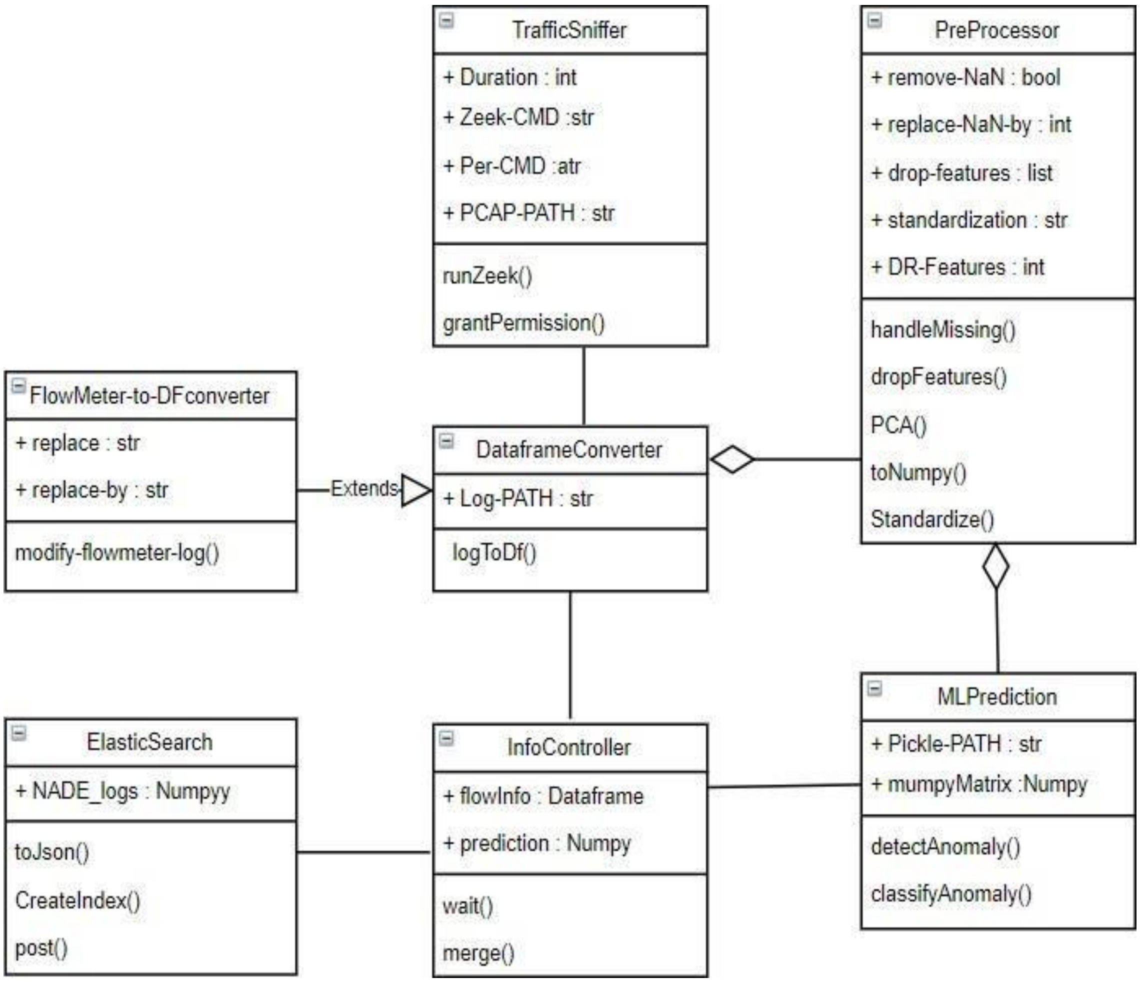
5.2.2.1 Deployment Diagram



5.2.2.2 Flow Diagram

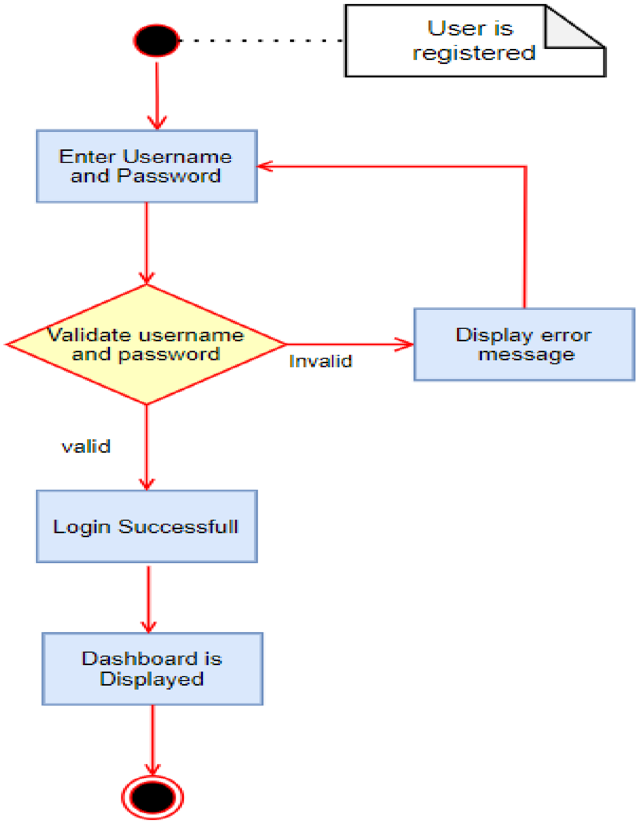


5.2.2.3 Class Diagram

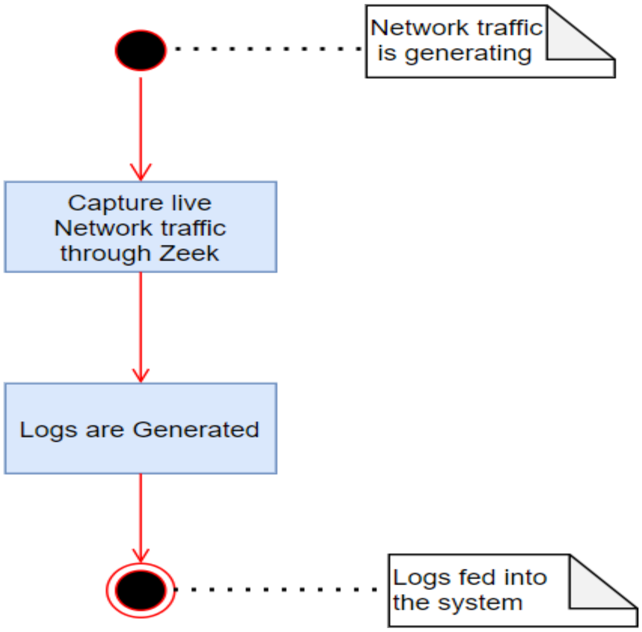


5.2.2.4 Activity Diagram

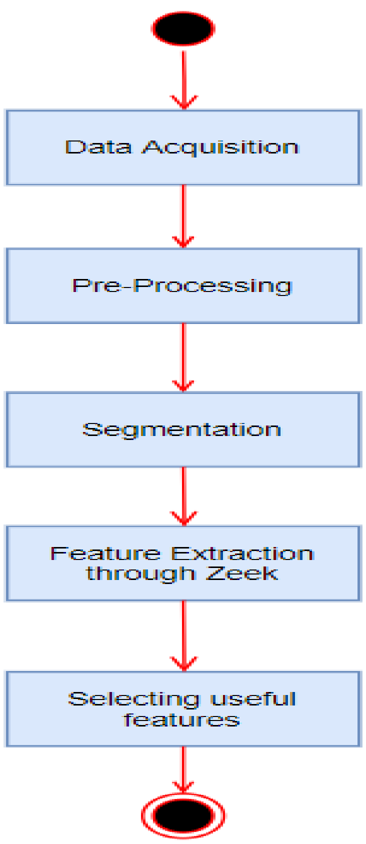
- Authentication



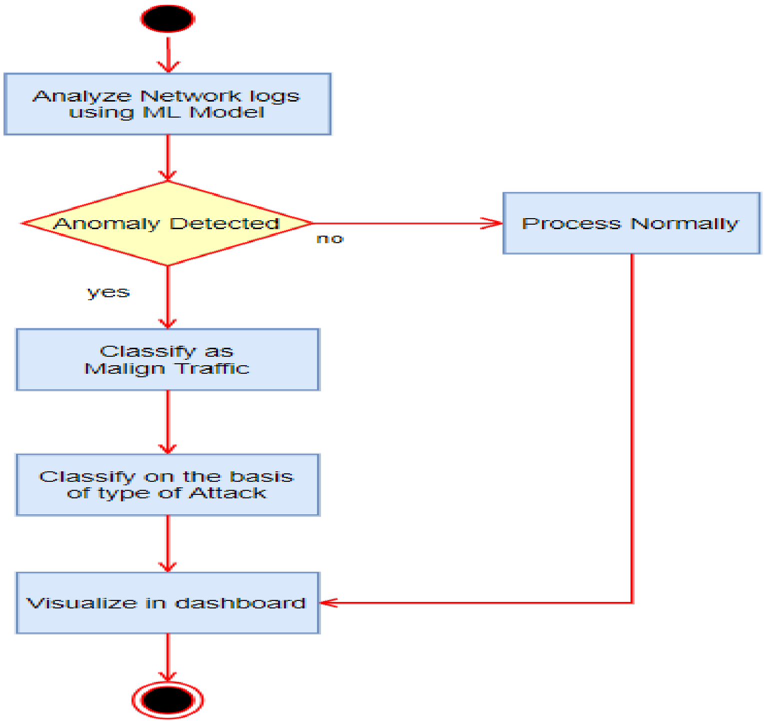
- Capture Live Network Traffic



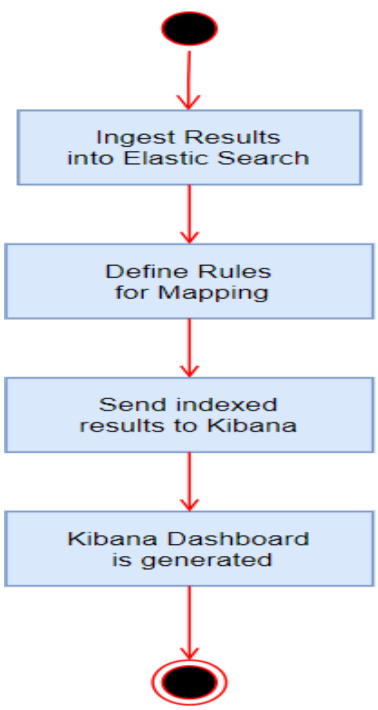
- Feature Extraction through Zeek



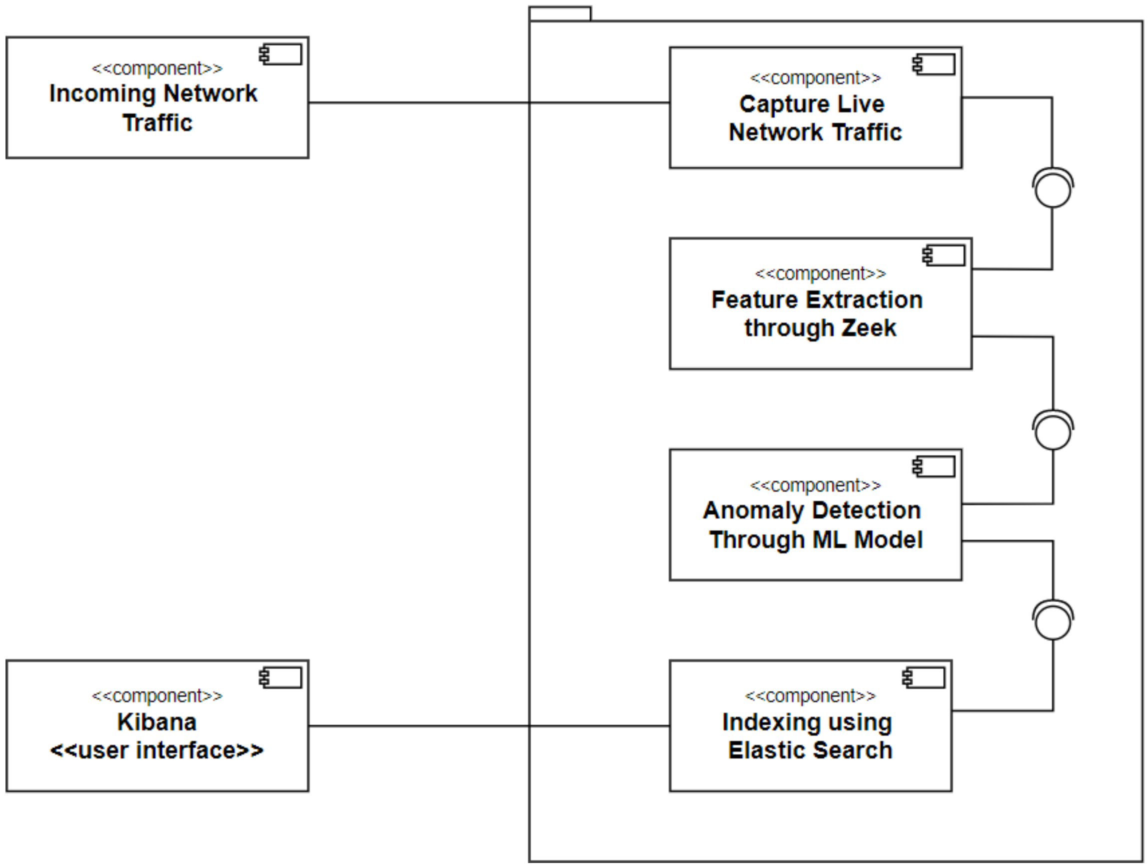
- Anomaly Detection using Machine Learning



- Data Indexing and Visualization



5.2.2.5 Component Diagram



5.2.3 Design Rationale

We have selected Model View Component Architecture(MVC). MVC patterns separate the input, processing, and output of an application. This model divided into three interconnected parts called the model, the view, and the controller. All the three above given components are built to handle some specific development aspects of any web or .net application development.

Following is some of the reasons for MVC selection:

1. We have considered this because we have 3 components and we want to segregate all three components, due to obvious security reasons, deployment of model is on the server end, which in this case is Model and other 2 components are on client side.
2. The requirement for data is to be processed fast, MVC fulfils this required by providing the asynchronous design.
3. Individual components require continuous modification of Model (ML MODEL) as to increase the efficiency and better scale to satisfy the modern needs.

4. The modifications on the Model does not affect the other components, hence clients will not face any disruption.

5.3 Data Design

5.3.1 Data Description

The mechanism/phenomena of data storage and information domain is very simple, the network traffic will be captured using Zeek's scripts and useful features will be extracted from network traffic on which the Machine Learning Model will learn.

The live network traffic is passed to the model and results are stored in the Elastic Search, which is based on NOSQL/Non structured DB.

5.3.2 Data Dictionary

Field	**Type**	**Description**
Forward Inter Arrival Time	Time Stamp	Time between two packets in forward direction.
Backward Inter Arrival Time	Time Stamp	Time between two packets in backward direction.
Flow Inter Arrival Time	Time Stamp	Time between two packets in either direction.
Active	Integer	Duration of sending packets for before going idle.
Idle	Integer	Duration of idleness before starting to send packets again.
Flow Bytes per second	Integer	Bytes per second in either direction
Flow Packets per second	Integer	Packets sent per second in either direction.
Duration	Integer	Time between the first and the last packet of the flow.

CHAPTER 6 : TESTING

6.1 TEST CASE # 1

Test Case ID: Fun_1
Test Priority (Low/Medium/High): Mid
Module Name: Kibana User (Any)
Test Title: Login
Description: Login with incorrect username or password

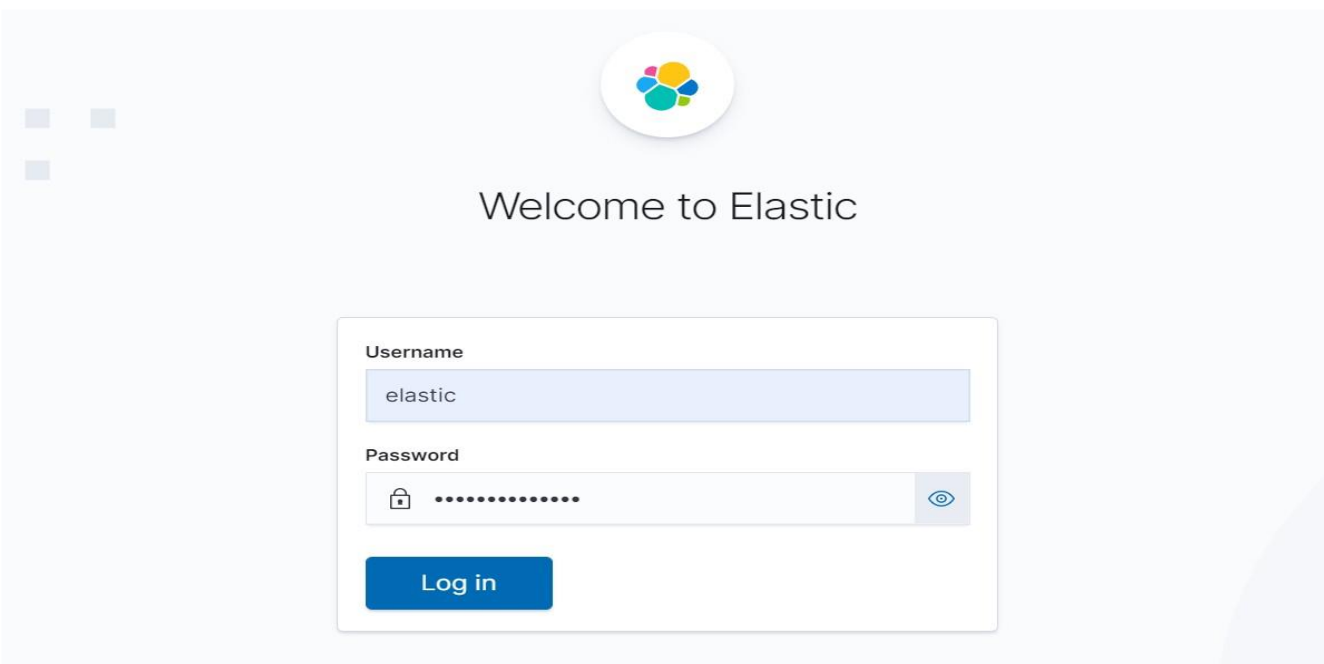
Test Designed by: Maryam Shafique
Test Designed date: 14-06-21
Test Executed by: Maryam Shafique
Test Execution date: 14-06-21

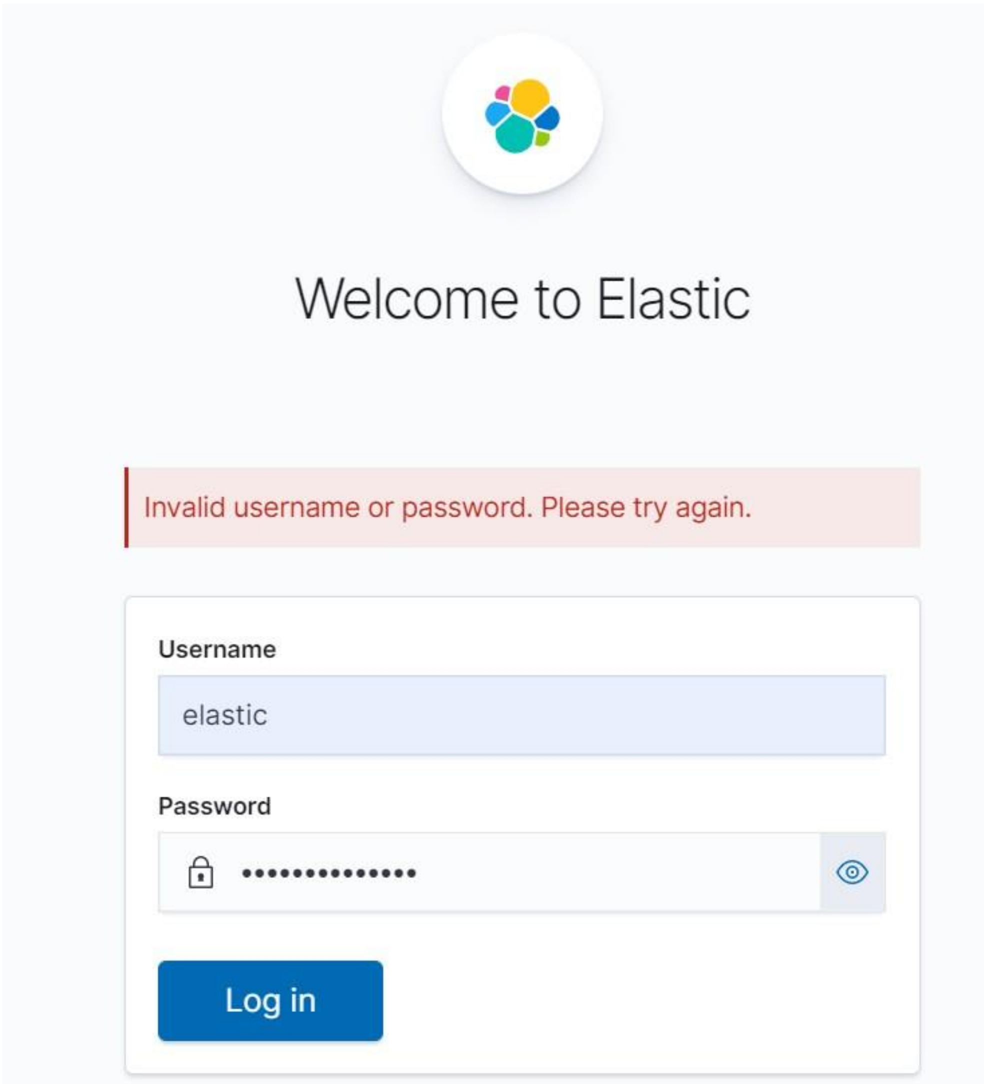
Pre-conditions: Login Page
Dependencies:

Step	Test Steps	Test Data	Expected Result	Actual Result	Status (Pass/Fail)	Notes
1	Open Kibana and Elastic search	Password checks	Error Message shown "Username/ Password is incorrect"	Incorrect login test passed	Pass	Implementation is correct
2	Open Login page	Username checks	Login failed			
3						
4						

Post-conditions:
Login checks successful.

OUTPUT:




Welcome to Elastic
Invalid username or password. Please try again.
Username
elastic
Password
Log in

6.2 TEST CASE # 2

Test Case ID: Fun_2

Test Priority (Low/Medium/High): Mid

Module Name: Website User (Any)

Test Title: Login

Description: Login by entering correct username/password

Test Designed by: Maryam Shafique

Test Designed date: 14-06-21

Test Executed by: Maryam Shafique

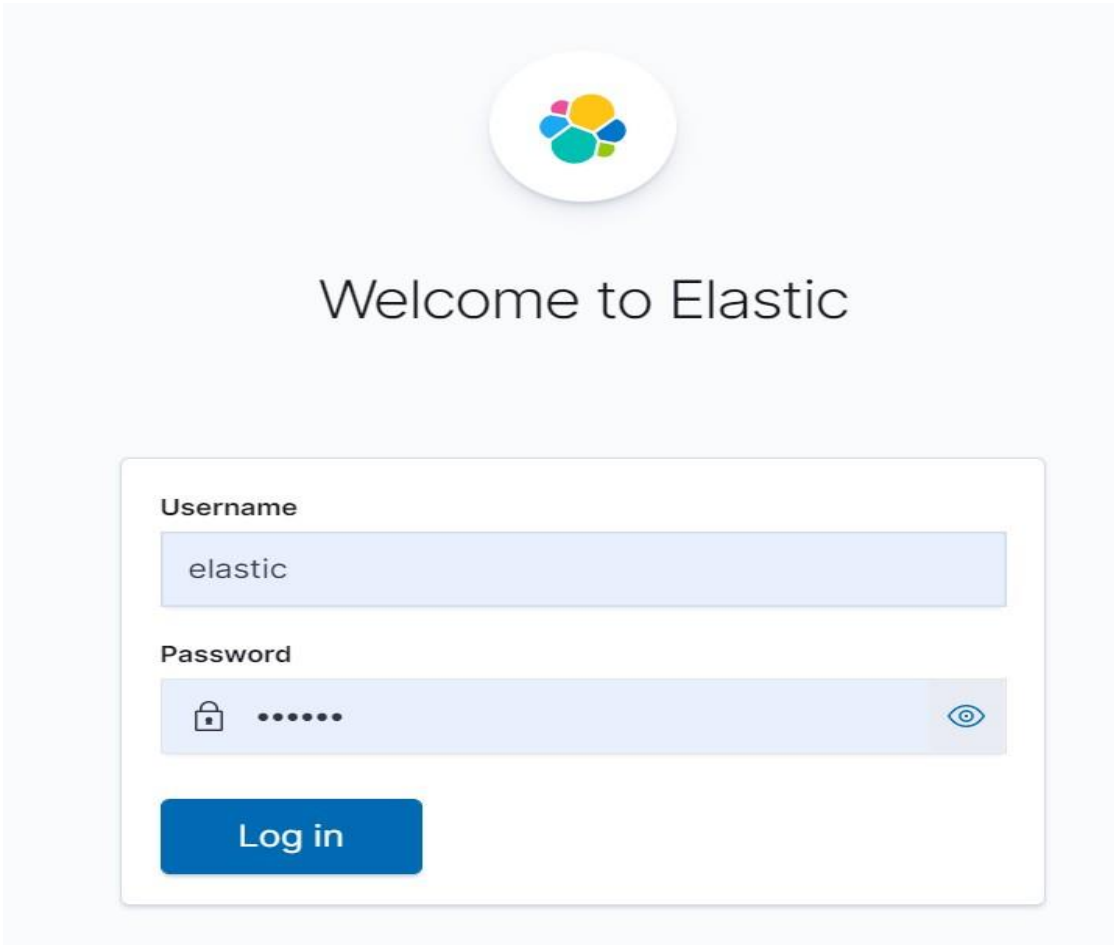
Test Execution date: 14-06-21

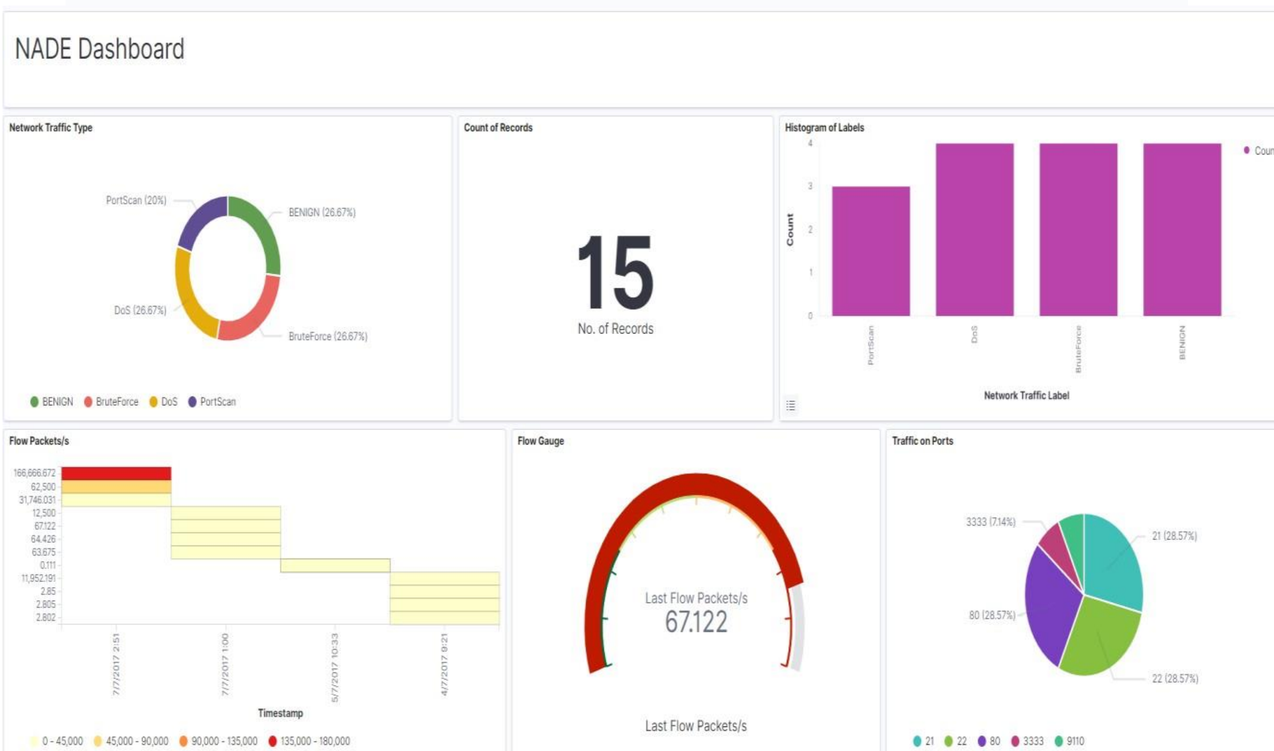
Pre-conditions: Login page
Dependencies:

Step	Test Steps	Test Data	Expected Result	Actual Result	Status (Pass/Fail)	Notes
1	Open elastic search and Kibana					
2	Enter valid username/ password	Entered username/Password	Login successful	Login successful	Pass	Implementation is correct
3	Click Login	Username/Password checks				
4						

Post-conditions:
Login checks successful.

OUTPUT:


Welcome to Elastic
Username
elastic
Password
Log in


NADE Dashboard
Network Traffic Type
PortScan (20%)
BENIGN (26.67%)
DoS (26.67%)
BruteForce (26.67%)
BENIGN BruteForce DoS PortScan
Count of Records
15
No. of Records
Histogram of Labels
Count
PortScan
DoS
BruteForce
BENIGN
Network Traffic Label
Flow Packets/s
Timestamp
0 - 45,000 45,000 - 90,000 90,000 - 135,000 135,000 - 180,000
Flow Gauge
Last Flow Packets/s
67.122
Last Flow Packets/s
Traffic on Ports
3333 (7.14%)
21 (28.57%)
80 (28.57%)
22 (28.57%)
21 22 80 3333 9110

6.3 TEST CASE # 3

Test Case ID: Fun_3
Test Priority (Low/Medium/High): Mid
Module Name: Kibana User (Any)
Test Title: Data Indexing
Description: Data indexing by Elastic Search

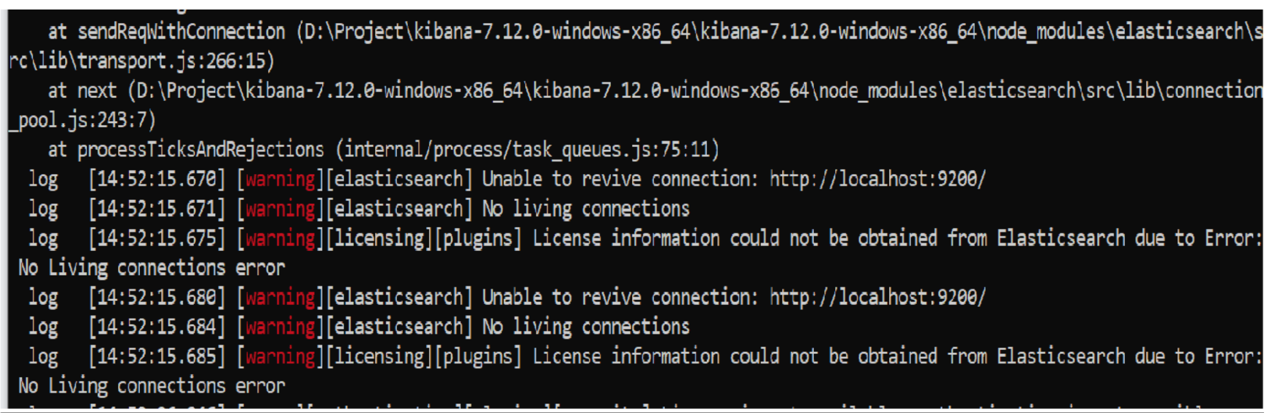
Test Designed by: Maryam Shafique
Test Designed date: 14-06-21
Test Executed by: Maryam Shafique
Test Execution date: 14-06-21

Pre-conditions: Elastic Search is not running
Dependencies:

Step	Test Steps	Test Data	Expected Result	Actual Result	Status (Pass/Fail)	Notes
1	Open Elastic Search					
2	Create index	Entered Data	Data indexing unsuccessful	Data Indexing unsuccessful	Pass	Implementation is correct
3	Ingest Data					
4						

Post-conditions:
Mapping is done correctly.
Data indexing not done. Index is not created.

OUTPUT:



```
    at sendReqWithConnection (D:\Project\kibana-7.12.0-windows-x86_64\kibana-7.12.0-windows-x86_64\node_modules\elasticsearch\s
rc\lib\transport.js:266:15)
    at next (D:\Project\kibana-7.12.0-windows-x86_64\kibana-7.12.0-windows-x86_64\node_modules\elasticsearch\src\lib\connection
_pool.js:243:7)
    at processTicksAndRejections (internal/process/task_queues.js:75:11)
  log   [14:52:15.670] [warning][elasticsearch] Unable to revive connection: http://localhost:9200/
  log   [14:52:15.671] [warning][elasticsearch] No living connections
  log   [14:52:15.675] [warning][licensing][plugins] License information could not be obtained from Elasticsearch due to Error:
 No Living connections error
  log   [14:52:15.680] [warning][elasticsearch] Unable to revive connection: http://localhost:9200/
  log   [14:52:15.684] [warning][elasticsearch] No living connections
  log   [14:52:15.685] [warning][licensing][plugins] License information could not be obtained from Elasticsearch due to Error:
 No Living connections error
```

{"statusCode":503,"error":"Service Unavailable","message":"License is not available."}

6.4 TEST CASE # 4

Test Case ID: Fun_4

Test Priority (Low/Medium/High): Mid

Module Name: Kibana User (Any)

Test Title: Data Indexing

Description: Successful Data indexing by Elastic Search

Test Designed by: Maryam Shafique

Test Designed date: 14-06-21

Test Executed by: Maryam Shafique

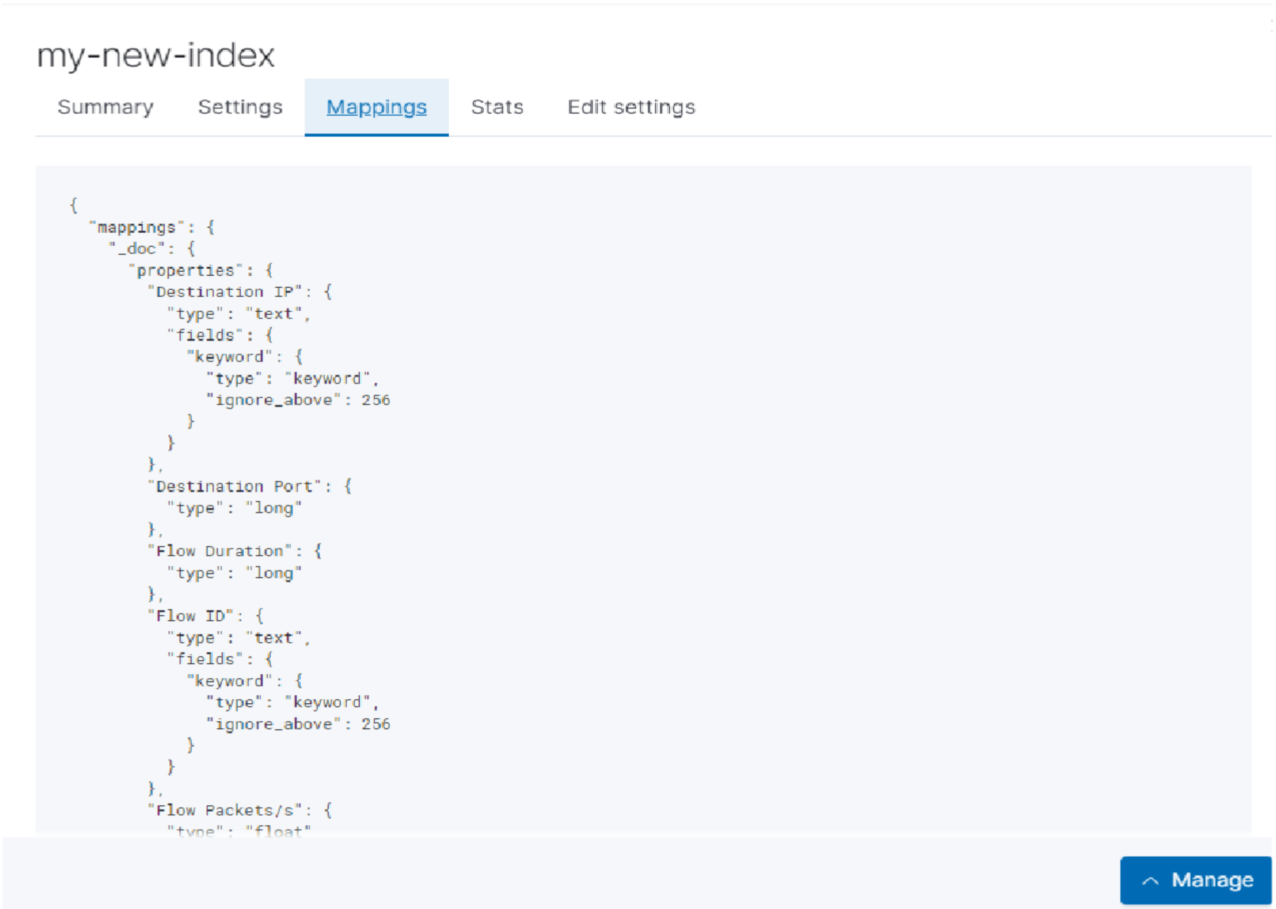
Test Execution date: 14-06-21

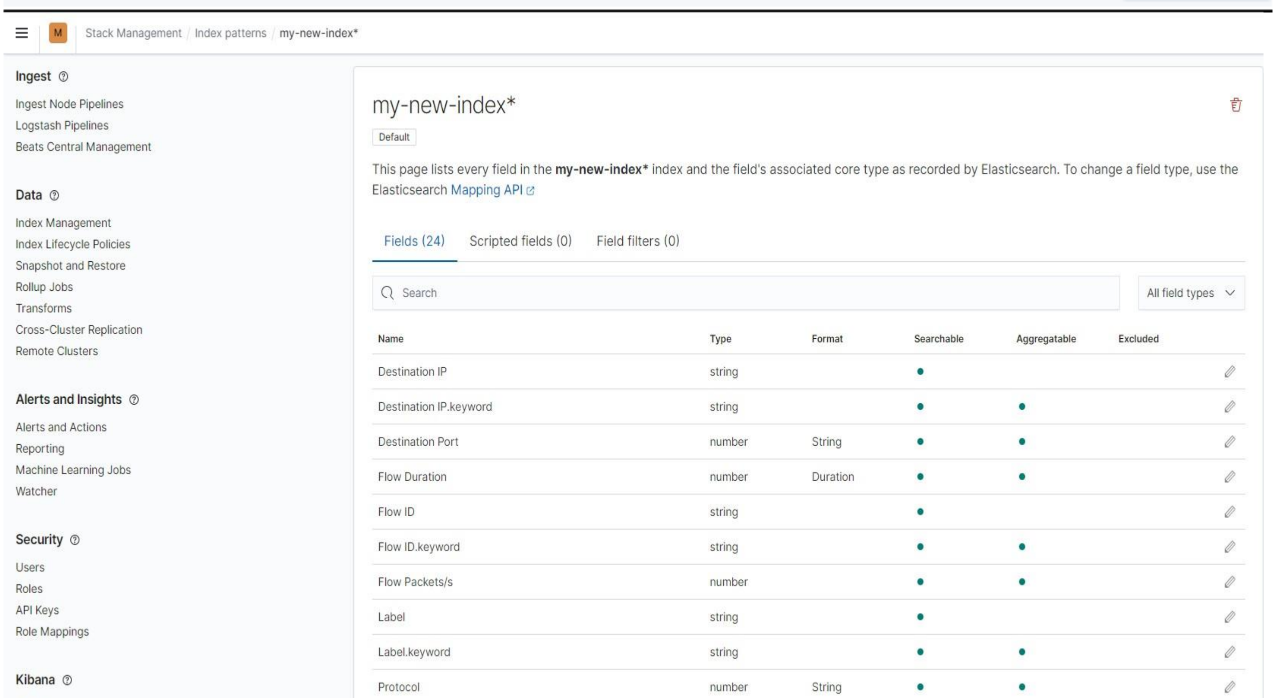
Pre-conditions: Elastic Search is running
Dependencies:

Step	Test Steps	Test Data	Expected Result	Actual Result	Status (Pass/Fail)	Notes
1	Open Elastic Search					
2	Ingest Data	Entered data	Data indexing successful	Data indexing successful	Pass	Implementation is correct
3	Create mappings					
4						

Post-conditions:
Mapping is done correctly.
Data indexing is done.

OUTPUT:


my-new-index
Summary
Settings
Mappings
Stats
Edit settings
{
"mappings": {
"_doc": {
"properties": {
"Destination IP": {
"type": "text",
"fields": {
"keyword": {
"type": "keyword",
"ignore_above": 256
}
}
},
"Destination Port": {
"type": "long"
},
"Flow Duration": {
"type": "long"
},
"Flow ID": {
"type": "text",
"fields": {
"keyword": {
"type": "keyword",
"ignore_above": 256
}
}
},
"Flow Packets/s": {
Manage


Stack Management / Index patterns / my-new-index*
Ingest
Ingest Node Pipelines
Logstash Pipelines
Beats Central Management
Data
Index Management
Index Lifecycle Policies
Snapshot and Restore
Rollup Jobs
Transforms
Cross-Cluster Replication
Remote Clusters
Alerts and Insights
Alerts and Actions
Reporting
Machine Learning Jobs
Watcher
Security
Users
Roles
API Keys
Role Mappings
Kibana
my-new-index*
Default
This page lists every field in the my-new-index* index and the field's associated core type as recorded by Elasticsearch. To change a field type, use the Elasticsearch Mapping API
Fields (24)
Scripted fields (0)
Field filters (0)
Search
All field types
Name
Type
Format
Searchable
Aggregatable
Excluded
Destination IP string
Destination IP.keyword string
Destination Port number String
Flow Duration number Duration
Flow ID string
Flow ID.keyword string
Flow Packets/s number
Label string
Label.keyword string
Protocol number String

6.5 TEST CASE # 5

Test Case ID: Fun_5
Test Priority (Low/Medium/High): Mid
Module Name: Kibana User (Any)
Test Title: Data Visualization
Description: Data is not visualized

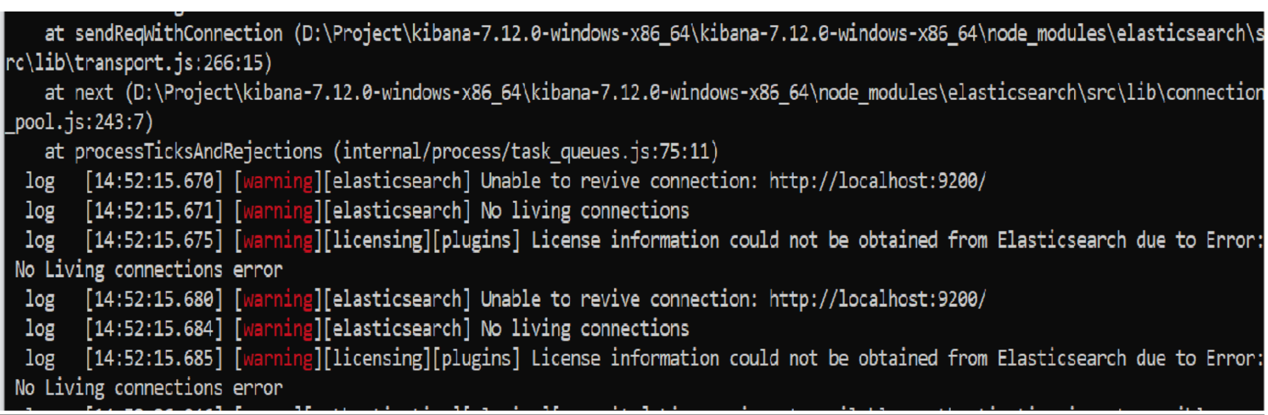
Test Designed by: Maryam Shafique
Test Designed date: 14-06-21
Test Executed by: Maryam Shafique
Test Execution date: 14-06-21

Pre-conditions: Elastic Search is not running.
Dependencies:

Step	Test Steps	Test Data	Expected Result	Actual Result	Status (Pass/Fail)	Notes
1	Open Elastic Search and Kibana					
2	Open Kibana Dashboard	Entered Data	Data is not visualized	Data is not visualized	Pass	Implementation is correct
3				Error message displayed		
4						

Post-conditions:
Kibana is running.
Data is sent to Kibana.
Elastic Search not running.

OUTPUT:



```
    at sendReqWithConnection (D:\Project\kibana-7.12.0-windows-x86_64\kibana-7.12.0-windows-x86_64\node_modules\elasticsearch\src\lib\transport.js:266:15)
    at next (D:\Project\kibana-7.12.0-windows-x86_64\kibana-7.12.0-windows-x86_64\node_modules\elasticsearch\src\lib\connection_pool.js:243:7)
    at processTicksAndRejections (internal/process/task_queues.js:75:11)
  log   [14:52:15.670] [warning][elasticsearch] Unable to revive connection: http://localhost:9200/
  log   [14:52:15.671] [warning][elasticsearch] No living connections
  log   [14:52:15.675] [warning][licensing][plugins] License information could not be obtained from Elasticsearch due to Error: No Living connections error
  log   [14:52:15.680] [warning][elasticsearch] Unable to revive connection: http://localhost:9200/
  log   [14:52:15.684] [warning][elasticsearch] No living connections
  log   [14:52:15.685] [warning][licensing][plugins] License information could not be obtained from Elasticsearch due to Error: No Living connections error
```

{"statusCode":503,"error":"Service Unavailable","message":"License is not available."}

6.6 TEST CASE # 6

Test Case ID: Fun_6
Test Priority (Low/Medium/High): High
Module Name: Kibana User (Any)
Test Title: Data Visualization
Description: Data Visualization Successful

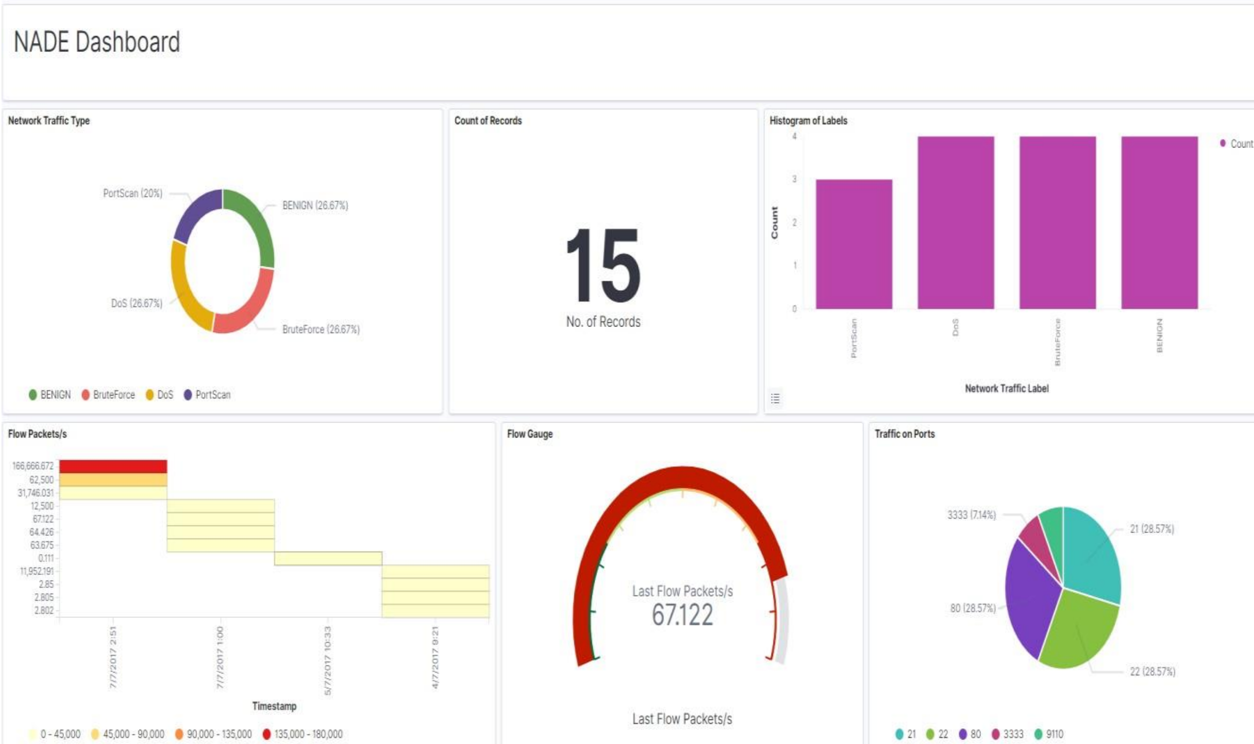
Test Designed by: Maryam Shafique
Test Designed date: 14-06-21
Test Executed by: Maryam Shafique
Test Execution date: 14-06-21

Pre-conditions: Elastic Search and Kibana are running
Dependencies:

Step	Test Steps	Test Data	Expected Result	Actual Result	Status (Pass/Fail)	Notes
1	Open website					
2	Enter valid email address	Entered email	Data is visualized successfully	Data is visualized successfully	Pass	Implementation is correct
3	Click Register	Regex for email validation				
4						

Post-conditions:
Kibana is running.
Data is sent to Kibana and visualized successfully.
Elastic Search is running.

OUTPUT:



Data Table

Timestamp	Flow ID	Source IP	Source Port	Destination IP	Destination Port	Label	Count of records
4/7/2017 9:21	172.16.0.1-192.168.10.5...	172.16.0.1	52350	192.168.10.50	21	BruteForce	1
4/7/2017 9:21	172.16.0.1-192.168.10.5...	172.16.0.1	52366	192.168.10.50	21	BruteForce	1
4/7/2017 9:21	172.16.0.1-192.168.10.5...	172.16.0.1	52368	192.168.10.50	21	BruteForce	1
4/7/2017 9:21	172.16.0.1-192.168.10.5...	172.16.0.1	52370	192.168.10.50	21	BruteForce	1
5/7/2017 10:33	172.16.0.1-192.168.10.5...	172.16.0.1	50600	192.168.10.50	80	DoS	1
5/7/2017 10:33	172.16.0.1-192.168.10.5...	172.16.0.1	50602	192.168.10.50	80	DoS	1
5/7/2017 10:33	172.16.0.1-192.168.10.5...	172.16.0.1	50604	192.168.10.50	80	DoS	1
5/7/2017 10:33	172.16.0.1-192.168.10.5...	172.16.0.1	50606	192.168.10.50	80	DoS	1
7/7/2017 1:00	192.168.10.12-192.168....	192.168.1...	35396	192.168.10.50	22	BENIGN	2

FUTURE WORK:

Project can be extended by incorporating alerts in Kibana. Alert rules can be defined to trigger an alert if attack is detected. Secondly, other attacks dataset can be generated and concatenated with current dataset or minority CIC dataset attack classes, which we have dropped for simplicity, can be up sampled. Generating other attacks dataset will increase the diversity of attack types and unknown attacks will also be detected. Other machine learning techniques should also be explored.

REFERENCES:

[1] Vilhelm Gustavsson, "*Machine Learning for a Network-based Intrusion Detection System An application using Zeek and the CCIDS2017 dataset.*"

[2] Ranjit Panigrahi, Samarjeet Borah, "*A detailed analysis of CICIDS2017 dataset ford esigning Intrusion Detection Systems*"

[3] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani, "T*oward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization*", 4th International Conference on Information Systems Security and Privacy (ICISSP), Portugal, January 2018

[4] Ahmed Ahmim, Leandros Maglaras, Mohamed Amine Ferrag, Makhlouf Derdour, Helge Janicke, "*A Novel Hierarchical Intrusion Detection System based on Decision Tree and Rules-based Models.*"

[5] CICIDS 2017 Dataset, *www.unb.ca/cic/datasets/ids-2017.html*

[6] CICFlowmeter, *www.github.com/CanadianInstituteForCybersecurity/CICFlowMeter*

[7] Zeek Script, *"www.github.com/zeek-flowmeter/zeek-flowmeter"*

[8] Joanne Peng*, "An Introduction to Logistic Regression Analysis and Reporting"*

[9] G. James, D. Witten, T. Hastie, and R. Tibshirani, An Introduction to Statistical Learning, 1:st ed. Springer, 2017, ISBN:978-1-4614-7137-0.

[10] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. 2016. [Online]. Available: http://www.deeplearningbook.org

[11] Tadeusz Pietraszek and Axel Tanner IBM Zurich Research Laboratory, Säumerstrasse 4, 8803 Rüschlikon, Switzerland, "Data Mining and Machine Learning—Towards Reducing False Positives in Intrusion Detection

PLAGIARISM REPORT