OPEN ARCHITECTURE BASED AUTONOMOUS CYBER AI FOR ANOMALY DETECTION

B.Sc. (Hons) in Information Technology
Specializing in Cyber Security

Department of Computer System Engineering

Sri Lanka Institute of Information Technology
Sri Lanka

October 2021

OPEN ARCHITECTURE BASED AUTONOMOUS CYBER AI FOR ANOMALY DETECTION

Dissertation submitted in partial fulfillment of the requirement for the Bachelor of Science Honors in Information Technology Specializing in Cyber Security

Department of Computer System Engineering

Sri Lanka Institute of Information Technology
Sri Lanka

October 2021

Declaration

We declare that this is our own work and this proposal does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any other university or Institute of higher learning and to the best of our knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

Name	Student ID	Signature
Hussain M.H	IT18166934	
De Silva H.W.D.T	IT18361728	
Madhuwantha K.A.N	IT18175530	
Liyanage U.I.D	IT18036626	

The above candidate is carrying out research for the undergraduate Dissertation under my supervision.

Signature of the supervisor Date

…… ……...…….

Abstract

With the rapid advancement in cyber threats and malicious activities in the current era along with the advancement in technology, available Intrusion Detection systems (IDS) are lacking in performance to identify such cyber threats and defend against novel attacks. Since most commercial IDS are based and relying on known signatures, it does not have the ability to detect zero-day or advanced malicious activities. In order to overcome the issue with signature-based IDS, a possible solution is to adopt anomaly-based detections to identify the latest cyber threats including zero days. Since the digital world is highly advanced, our focus is initially on network intrusions. There are current solutions that work with behavior-based anomaly detection. This research paper discusses detecting network anomalies using AI-based technologies such as machine learning (ML) and natural language processing (NLP). In the proposed solution, network traffic logs and HTTP traffic data are taken as inputs using a mechanism called beats. Once relevant data has been extracted from the captured traffic, it will be passed to the AI engine to conduct further analysis. Algorithms such as Word2vec, Convolution Neural Network (CNN), Artificial Neural networks (ANN), and autoencoders are used in order to conduct the threat analysis. HTTP DATASET CSIC 2010, that NSL-KDD, CICIDS are the benchmarking datasets used in parallel with the above algorithms in order to receive high accuracy in detection. The outputted data is integrated and visualized using the Kibana dashboard and blockchain model is implemented to maintain and handle all the data.

Index Terms- NLP, Anomaly detection, Network Intrusion Detection, Deep learning, word2vec, ANN, CNN, Beats

Acknowledgment

We would like to express my special thanks of gratitude to our supervisor Ms. Chethana Liyanapathirana for accepting us and guiding us in this project till the very end.

We would also like to express my thanks to my Co-supervisor Dr.Lakmal Rpasinghe for his extended support, guidance and motivation in making this project a success.

We would also thank the coordinator of the RP module Dr. Janaka Wijekoon for sharing the proper knowledge to conduct this research and also thanks to all our industrial colleague for their support in our difficult times. Finally, we thank all our friends to their valuable support and sharing knowledge to the success of this research.

Table of Contents

List of figures

List of tables

List of Abbreviation

ABBREVIATION	DESCRIPTION
AI	Artificial Intelligence
IDS	Intrusion detection system
NIDS	Network intrusion detection system
HIDS	Host based intrusion detection system
DPI	Deep packet Inspection
ML	Machine learning
DL	Deep learning
NLP	Natural Language processing

1. Introduction

With the development of technology, computer-supported crimes are also exponentially evolved. Besides the aforementioned large number of attack surfaces, one of the major reasons for that is unlike earlier nowadays people who have less technical knowledge like script kiddies also can perform malicious activities due to the highly sophisticated tools that are available in the world which require less expert knowledge as they used to automate the attacking processes.

Despite there are experts like black hat hackers and grey hat hackers who are sponsored to perform cybercrimes, professionals who are securing the computer networks against cybercrimes cannot deny the fact that script kiddies also could perform significant damages to computers and networks without even knowing due to the capabilities of highly sophisticated tools. Therefore, securing a particular computer network or any computer which is connected to LAN or WAN against cyber-attacks has become a very challenging task.

Moreover, due to these reasons a huge market space has been created to cyber security industry to be observant enough to protect this information from malicious activities. Most of the time the attackers are always a step ahead of the security analysts, so that proactive measures are to be taken to protect the systems from intruders.

To address these problems, we have proposed an autonomous cyber artificial intelligence system to capture the anomalies from the network traffic and notify to the users to take suitable countermeasures before any harm happens. Since that intruder cannot be ignored, there should be a strong intrusion detection system and react to it proactively. The final output of the product is meant to be an SIEM (Security Incident and Event Management) therefore, an AI engine and a neuro realistic engine will be used to analyze all the data that has been captured from the network and the host systems by using a mechanism call beats where the captured data will be forwarded to the AI engine and neuro realistic detection for further analysis. For a broader output

of the system, we have to focus both on network intrusion detection system and as well as host-based detection system, where ELK engine will be used to capture the host-based data and zeek will be used to capture the network-based data. The proposed system of ours train both NLP and ANN models using latest datasets which contain novel attacks instances which will lead to provision robust defense against novel network attacks and the improvement of the accuracy and the reliability of the trained model.

In order to conduct the threat analysis, the proposed deep learning and machine learning models are Algorithms such as Word2vec, Convolution Neural Network (CNN), Artificial Neural networks (ANN), and autoencoders are used in order to conduct the threat analysis. autoencoders are used in order to conduct the threat analysis. HTTP DATASET CSIC 2010, that NSL-KDD, CICIDS are the benchmarking datasets used in parallel with the above algorithms in order to receive high accuracy in detection.

When it comes to storing the data securely, storing data in isolation is the most common approach taken current era where only authorized people are allowed to access. As an advanced solution for this traditional approach, we will be using the blockchain mechanism to store the captured data and the processed output where separate blocks will be used to store the data, not only that each block will be composed with a time stamp, a cryptographic hash, a parent block and a nonce value to ensure the safety of the stored network traffic and the processed output.

1.2 Background Literature

With the advancement in technology which has made huge advances in packet capture and has designed both software and hardware solutions to capture packets across the network. This research focusses on network data and not host data captures. And it also focuses on the security of the network where the data transmitted over the communication medium should be well secured since there is a possibility of man in the middle attack where the hacker could obtain the data and re-insert a false message [1].it is needed to make sure that data is being collected from various sources for network monitoring. Meanwhile there are possibilities in attacks when transmitting data over network, therefore choosing an efficient network monitoring application is critical [1]. The authors have found that DAG technology is used as a hardware solution since it can capture 100G of packet size but due to its excessive cost compared to software solution, it is considered inefficient [1].

In this research the author has mentioned that by focusing on capturing data the identification of well-known ports and DPI (deep packet inspection) are the mainstays of conventional approaches to network traffic classification. However, there are many new cyber threats that could easily bypass these kinds of traditional methods and since the use of HTTPS governments are promoting laws to prevent organizations in inspecting data. Due to these factors there are new analysis methods using ML (machine learning) techniques that has given industrial users the thirst to research for it [2].

In another research it is believed that Despite the effort on developing IDS, there are still issues faced on improving detection accuracy and while reducing the false alarm rates in detecting threats. As a result, researchers have found solutions with the help of Machine learning (ML) and Deep learning (DL) techniques to design the Intrusion detection system (IDS) [3]. The concept in this research for capturing data is to mirror all incoming and outgoing network traffic for the intrusion detection system to perform network traffic. The authors also further have discussed about the research challenges for the IDS which talks about the low accuracy in detection due to imbalance data and the low performance in the real-world systems since high processing power is needed to run such mechanisms like IDS which is designed with ML and DL

techniques. Further this research discusses on the future trends in the scope of IDS where the researches talk about detecting zero-day attacks with the use of proper and balanced data and it provides a solution for complex models [3].

The authors have discussed [4] these IDS which are build based on machine learning methods can be divided into three main categories based on the machine learning methods that are used in those systems. The first one is called as supervised model which is trained to learn from labeled data of both normal traffic and anomalous traffic instances whereas the second method called as semi-supervised ML model in which the model is trained using a small amount of labeled data with a considerably large amount of unlabeled data. The third one is the unsupervised ML method which uses clustering techniques to classify unlabeled data. Most of the systems are occupied with this unsupervised machine learning model due to its ability to detect anomalies of the existing network and the ability to detect zero-day attacks. But there are couple of drawbacks in machine learning when it compared with deep learning techniques employed in new intrusion detection systems, like machine learning based systems doesn't have the capabilities to go through the obstacles and adapt to them accordingly while performing intended task, inability of provisioning robust solutions for complex scenarios.

When it comes to deep learning techniques Ali et al study [9] proposed an approach based on fast learning network and based on particle swarm optimization method based on artificial neural networks and they comprise input layer, output layer and multiple hidden nodes which can mimic the behavior of human brain. They demonstrated that the number of neurons that are there in the hidden layers of the deep learning model could affect to increase the accuracy. Although it outperformed when compared to other related work it was another supervised based approach which had some common drawbacks like misclassifying unknown traffic into previously trained classes, requirement of large amount of labeled data to train the model and inability to discover the features of given data and cluster them on its own. Al- Qatf et al [11] study proposed a mechanism in which the researchers utilized sparse autoencoders and support vector machine and it was an unsupervised approach which could detect the features of the network traffic without labeled data and correctly

classify anomalous traffic which was previously unknown to the proposed system also it was able to overcome number of drawbacks that were there in supervised based deep learning approaches but there were some drawbacks as well.

Because of that available IDSs such as firewalls, anti-viruses are not capable of detecting advanced malicious activities [3]. According to PandaLabs annual report of 2018 [4], "in 2018 continues to reflect the prevalence of malware attacks, with 9 million malicious URLs and 2.4 million attacks blocked per million endpoints per month. 20.7% of machines studied experienced at least one malware attack during the period analyzed". Figure 1.1.2 shows the escalated malware incidents to PandaLabs within four years. Most of these attacks happened because available security products could not detect those attacks effectively. Widely used Intrusion Detection Systems (IDS) uses signature-based [5], detection technologies, and methods using blacklists. If the attackers change the patterns of their methods or change blacklisted malicious destinations, security products cannot detect those attacks [6]. Even though those systems detect malicious activities effectively for existing known attacks, no protection from newly created attacks like zero-day attacks and improved attacks.

To address that problem, this paper proposed to use an anomaly-based intrusion detection system. An anomaly detection system identifies malicious activities early by comparing abnormal and normal behaviors of the system after training a model [7]. This model-based detection approach is able to address both known and novel attacks. There are two main types of IDSs known as Host intrusion detection systems (HIDS), and the second one is the Network intrusion detection system (NIDS) [1]. As most cyber-attacks happened through the network, I firstly focused on network anomaly detection. Network anomaly detection systems are mainly detecting anomalies from network traffics by analyzing behaviors. Detecting unknown malicious traffic is a challenging task.[8] Only use behavior analysis will not give effective results due to malicious attacks improving along with the detection systems, and there are probabilities to attacks hide under normal behavior. Therefore, this paper discusses about detecting anomalous network traffic using Natural Language Processing (NLP) [9]. Since NLP focused on the textual and non-textual data, using semantic and syntactic analyzing. It will be able to identify related words, instances,

and features of a particular attack. In this approach, I trained two deep learning models to identify a particular attack's nature and classify network traffics and the URLs as either abnormal or benign.

Under this section of research, we mainly focus on storing the captured data in the most secure way. As this is an AI based anomaly detection system, we are planning to use Artificial Intelligence for storing the data securely because it is a manner to organize and use the data very efficiently and also is elevated the execution of complex programs [1] In this research the proposed solution is to encrypt the captured data and then store it in a blockchain where it is AI enabled which will help to organise and use the data in more orderly and efficient way. And also, when it comes to zero-day attacks and novel attacks we are proposing a way to store the data in a separate dataset for those attacks and store them in the blockchain.[5]

By design blockchains are decentralised, secure, immutable and extremely fault tolerance making them the best suitable way to store the data or information which his at rest.[3]

Each and every record of an event is cryptographically secured with the help of a digital signature, which is ordered, verified and bundled together into blocks which forms the transactions in blockchains.[3] You might still have a question. How can the blockchain transaction or storing data in a blockchain is secure? Each block is composed with transaction data along with a timestamp, cryptographic hash of the previous block which is the parent block and a nonce (random number or a but string used to verify the hash)

The security and the immutability of the transaction being stored on the block is ensured by Secure Hash Algorithm (SHA) which also makes sure that it has authenticity and integrity. When we talk about algorithms blockchain uses the Consensus Algorithms which will help to achieve the agreement on a single state of data in a distributed network, ensures that it helps prevent the malicious nodes from changing the state of the data

1.3 Research Gap

Currently there are several monitoring system platforms which provide which has its own specifications and capabilities up to a certain level.

In terms of the current solutions, they focus on providing limited features while doing the monitoring [1]-[3]. For example, Wireshark is one of the leading network monitoring tools, it provides rich content to users, but it fails to provide geo locations of the traffic. Likewise, there are some drawbacks in good platforms as well. There is also a need for proper enrichment of these captured data. These systems forward all the data that has been captured for further analysis enrich may take time in the analyzing process.

And these are some concerns to be taken in providing an efficient and effecting monitoring systems. The current prevailing intrusion detection systems are built using signature-based detection mechanisms also known as misuse mechanisms and therefore the networks that use those intrusion detection systems are vulnerable to zero-day attacks and, they are only capable of detecting previously known attacks which basically omits detecting known attacks if their signature databases didn't get updated frequently. However, because of the complex nature of computer networks, sophisticated novel attacks, usage of adversarial AI technologies, the exponential growth of data requires domain specific expertise to defend against cybercrimes.

Another problem that has been found is that the systems that enabled deep learning mechanisms tend to overwhelm the computer resources and require more computational power and take considerably large amount of time to train the model. Though there are IDS systems which were developed based on machine learning techniques some of them used supervised machine learning mechanisms which again, result to detect only trained attack types. Another problem is that most of the IDS used unsupervised machine learning mechanisms trained using well known data sets like KDD99 and DARPA which were created about two decades ago, and which causes to IDS to misclassify highly sophisticated attacks as normal traffic or completely bypass the IDS system without detecting by it.

There are so many ways to store the captured host data securely and in an efficient way. The main points that taken into consideration is. How to store the captured data in an effective and an orderly manner? Ensuring the security of the stored data and Hashing the data before saving inside the blockchain

In the Autonomous cyber-AI for Anomaly data capture mechanism monitoring is designed in a way which provides more enhanced and enriched data compared to the existing solutions available. In my proposed solution, it has been designed to focus on both network and host data and to provide much enriched and enhanced data and to clearly define all the fields and make sure that 100% legitimate data are not forwarded to the AI engine and moreover log stash is used to filter all the captured traffic and clearly define fields so that any of the captured data are not dropped while forwarding it to the AI engine and resulting. It would be convenient for the AI engine for further deep analysis. The proposed blockchain is designed with much more data storage capabilities and it comes with more security that the prevailing Anomaly Detection systems also we are planning of hashing the gathered data before storing in the blockchain so that we can save lots of storage space and also it will be easily when retrieving the data because the data can be searched using it hash value.

1.4 Research Problem

For intrusion detection systems to be accurate, the data feeding into it should be meaningful, rich, and enhanced so that it could detect even zero-day attacks using AI apart from pre-defined signature-based attacks. Therefore, capturing network traffic in real time and dropping of important packets from network which may contain malicious content like virus or worm may affect the network [1]. Likewise in collecting data from hast based, it is important to clearly filter the data so that is would be easy for AI to conduct the further analysis. To achieve accurate decisions from IDS, there should be a proper data classification and proper data feed into the system. Furthermore, implementing network intrusion detection system not that much easy process. Most researchers try to implement intrusion detection systems to secure devices from intruders. Since the cyber threats are being improved along with the advancement in intrusion detection technology, available security products (Firewall, sandbox, virus guard, SIEM) are no longer cable to protect systems from improved and novel attacks. Because most of the legacy systems are relayed on predefined rules and signatures.

On the other hand, anomaly-based IDS solutions are coming handy and highly effective when it comes to detecting previously known attacks as well as unknown threats based on anomalous behavior of network traffics but unfortunately adversarial AI is highly taking place in which the perpetrator tends to obtain some sort of knowledge about defensive ML models that are deployed in computer networks and then try to fool the ML models by providing deceptive input using adversarial AI techniques which cause to malfunction the defensive ML technologies also there are some drawbacks of the promising accuracy levels due to the high rate of false positives and false negatives of anomaly based IDS solutions.

Apart from that data at rest has the highest value our sole purpose is to secure the data from the attackers. With the advancement of the technologies used in storing data securely has become advanced as well as very much challenging. With the attackers thinking a step Infront us makes our data and information vulnerable without even our knowledge. The current ways of storing all the data that is gathered from the host and

networks are mostly outdated and have no previous gathered data or information about zero-day attacks and novel attack, which makes the data at rest very vulnerable. As most of the datasets being outdated the risk level rises against novel attacks. And, as there is no way to orderly sort out the data the blockchain becomes flooded with useless information. For the data storing to be secure and much more efficient the data that is stored should be analyzed and sorted accordingly before storing in. When this happens the security of the stored data goes down. Hence is a high demand of a block chain which captures and store all the captured data without any analysis or classification. But when this happens the storage problem arises as thousands of data are being captured daily because all the captured data gets stored in the blockchain which will be a huge factor in the security of the stored data in the blockchain. But still even if the data that is captured is filtered and sorted there will be a good amount of data that will be stored within the blockchain. In the long run the storage problem will come into play so we should propose a method to fix the size of the data to a fixed amount before being stored. And when retrieving the stored data, the process should not take forever as this is an anomaly detection system.

Moreover, Under the lens of chaos engineering, manually configuring IDS system is ineffective because it disables system's capability to withstand against turbulent and unexpected conditions. Therefore, in this research it is focused to build an open-source product called “Autonomous Cyber AI” which can address aforementioned problems and for detecting network intrusions and anomalies by eliminating existing drawbacks of the novel systems which can deal with the complexity of the computer networks and capable of provisioning robust security in an efficient, highly accurate and cost-effective manner.

2. Research Objectives

For the past couple of decades, the world has witnessed a revolutionary change in information technology along with IoT and computer networks which enable people to control, automate things, make accurate decisions easier and it has affected humans' day-to-day lives also. These technologies have invaded almost every field like manufacturing and industrial, agriculture, energy, poultry and farming, transportation, healthcare, and many more not only that, but also from smartphones and laptops it is growing up towards connected cars, connected wearables like smart watches, fitness trackers, smart houses and concepts like smart cities. These technologies make humans' lives easier by reducing human efforts and saving a considerable amount of time in daily life. Even though there are so many more benefits to mankind, the connected and integrated technology of these devices and computer networks, paves way for new attack vectors for adversaries due to the large number of attack surfaces. These attacks can vary from single person's credential harvesting by simply tricking them to provide their login credentials to compromising computer networks of critical infrastructures such as baking, communication, electricity, healthcare, and transportation in which the disruption, destruction could have a deliberating impact on the nation's wellbeing and security.

Because of the development of technology, computer-supported crimes are also exponentially evolved. Besides the aforementioned large number of attack surfaces, one of the major reasons for that is unlike earlier nowadays people who have less technical knowledge like script kiddies also can perform malicious activities due to the highly sophisticated tools that are available in the world which require less expert knowledge as they used to automate the attacking processes. Despite there are experts like black hat hackers and grey hat hackers who are sponsored to perform cyber-crimes, professionals who are securing the computer networks against cybercrimes cannot deny the fact that script kiddies also could perform significant damages to computers and networks without even knowing due to the capabilities of highly sophisticated tools. Therefore, securing a particular computer network or any computer which is connected to LAN or WAN against cyber-attacks has become a very

challenging task. Moreover, due to these reasons a huge market space has been created to cyber security industry to be observant enough to protect this information from malicious activities. Most of the time the attackers are always a step ahead of the security analysts, so that proactive measures are to be taken to protect the systems from intruders. To address these problems, we have proposed an autonomous cyber artificial intelligence system to capture the anomalies from the network traffic and notify to the users to take suitable countermeasures before any harm happens. Since that intruder cannot be ignored, we must have a very strong intrusion detection system and react to it proactively. So, we will be using an AI engine to analyze all the data that has been captured from the network and the host systems by using natural language processing (NLP) where the captured data will be forwarded to the AI engine for further analysis. For a broader output of the system, we have to focus both on network intrusion detection system and as well as host-based detection system, where ELK engine will be used to capture the host-based data and zeek will be used to capture the network-based data. The proposed system of ours train both deep learning model and unsupervised machine learning model using latest datasets which contain novel attacks instances which will lead to provision robust defense against novel network attacks and the improvement of the accuracy and the reliability of the trained model. When it comes to storing the data securely, storing data in isolation is the most common approach taken current era where only authorized people are allowed to access. As an advanced solution for this traditional approach, we will be using the blockchain mechanism to store the captured data and the processed output where separate blocks will be used to store the data, not only that each block will be composed with a time stamp, a cryptographic hash, a parent block and a nonce value to ensure the safety of the stored network traffic and the processed output.

The result of this research is to develop a self-learning cyber-AI for detect and respond to newly created cyber-attacks in real time without human involvement. To achieve the main objective of the Autonomous Cyber AI we have divided our functions into four main parts.

- Implementation of Threat Detection Engine

- Implementing a Neuro Realistic Detection Engine
- Implementing AI Engine
- Implementing and Integration of a Secure Blockchain to store the data.

To achieve the main objective, the specific objectives that should be fulfilled are as follows.

1. To capture real time from the host and network for deep learning.
   Initially to deal with the huge network bandwidth and the speed of network traffic and to collect all host-based information including log files, files accesses and to be effective in capturing these we are using beats.

2. Ensure improved accuracy in threat detection by capturing all necessary data for threat detection.
   To produce a productive output in Autonomous Cyber AI, it is needed to make sure that we maintain high accuracy and reduce the rate of false alarms, the foundation to make a highly accurate detection system is in the hands of how data is captured and the type of data, therefore capturing all data from various sources of host and network to ensure the high accuracy in detection is to be implemented.

3. Find the best algorithm for word embedding.
   To classify network traffics as abnormal or normal, the model should identify words related to a particular attack. Therefore, in this phase, train several algorithms separately to achieve this objective and choose the best performant algorithm by comparing the results. Then continue training with that algorithm to increase accuracy.

4. Finding the best algorithm for word embedding
   To classify network traffics as abnormal or normal, the model should identify words related to a particular attack. Therefore, in this phase, train several

algorithms separately to achieve this objective and choose the best performant algorithm by comparing the results. Then continue training with that algorithm to increase accuracy.

5. Latest and most suitable bench marking datasets should be gathered.
   Training different machine learning models and deep learning models and compare the outcomes with previous research work and determine the most effective model after observing the experimental results.

6. Automating network data gathering
   Build a mechanism which automates the network data gathering process of the artificial intelligence engine. Since the proposed autonomous cyber-AI provides protection for the real time network anomalies all the network data stored in the Elasticsearch should be continuously retrieved and flawed to the proposed AI engine.

7. Using AI engine to analyze network data

Deploy the AI engine to analyze the retrieved network data in the previous phase and automate the network traffic classification process utilizing the artificial intelligence engine.

8. Orderly storing the captured in an orderly and an effective way in the blockchain

We should store the captured host-based information such as log files, files accesses and store them in an efficient way so to store it in an efficient way we use the blockchain. Also, blockchains are decentralised, secure, immutable, and extremely fault tolerence and with the AI is which will help to organise and use the data efficiently. Hashing the data before storing helps us to use the storage data in a more efficient way. As we are using blockchains to store the data which is gathered the security of the stored data will be at higher levels. Because each

block of the data will be composed with a timestamp, cryptographic hash and a nonce.

9. Merkle Root and Proof of Work

As we are capturing thousands of data and storing thousands of data on theblockchain it will consume lot of storage in the blockchain will be a problem in the long run. So, we are proposing the Merkle tree method which will lower hash levels, allowing for quicker verification and transactions. Working proof(PoW) is a blockchain network unique consensus algorithm. The algorithm is used to validate the transaction and to produce a new chain block. In this method, minors (a group of individuals) compete to complete the network transaction. Proof of work demands those in the network who own the computers to resolve a challenging arithmetic issue so that a block may be inserted into the network.

10. Implementing the blockchain in the cloud.

In distributed cloud storage, customers are connected to a secure P2P network, which enables decentralized storage without compromising security on their mission important data. Blockchain technology offers other strong features to keep your information. It is 10 times faster and the overall cost may be reduced to half. The strategic combination of distributed cloud storage includes openness and security and hash function, public / private key encryption, and secure transaction ledgers.

Decentralized cloud storage also safeguards customers from several security concerns. In addition, only with the aid of client-side encryption is available access to encryption keys and unencrypted data to the end user. All this allows customers to govern their various data assets completely.

3 Methodology

3.1 Methodology

3.1.1 System overview of the Autonomous Cyber AI

The Autonomous Cyber AI is meant to detect anomalies in an effective and efficient manner. To achieve high accuracy, this research focuses on four main components which comprise Threat intelligence Engine, AI engine, Neuro Realistic detection and secure blockchain model.

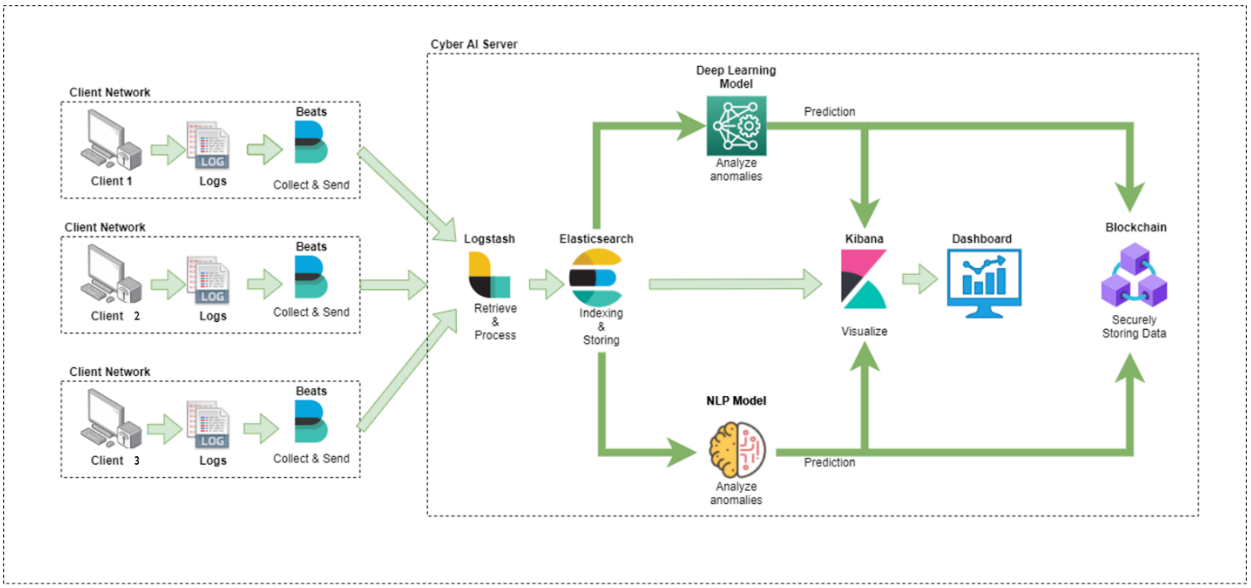


Figure 3.1.1.1 system overview of the autonomous cyber AI

3.1.2 Functioning and Implementation of threat engine

The proposed data capture engine has the capability of,

- Capture real time traffic and data from multiple sources.
- Process the captured data into a meaningful format.
- Enrich and enhance the captured information for further analysis.
- Identify legitimate traffic and discard them from further analysis.

This data capture engine is used to make accurate decisions on anomaly-based intrusions. And it collects data and network traffic real time from the host and the network. This system is focused to collect every data in the host system and network traffic making sure not to miss any important data for anomaly detection.

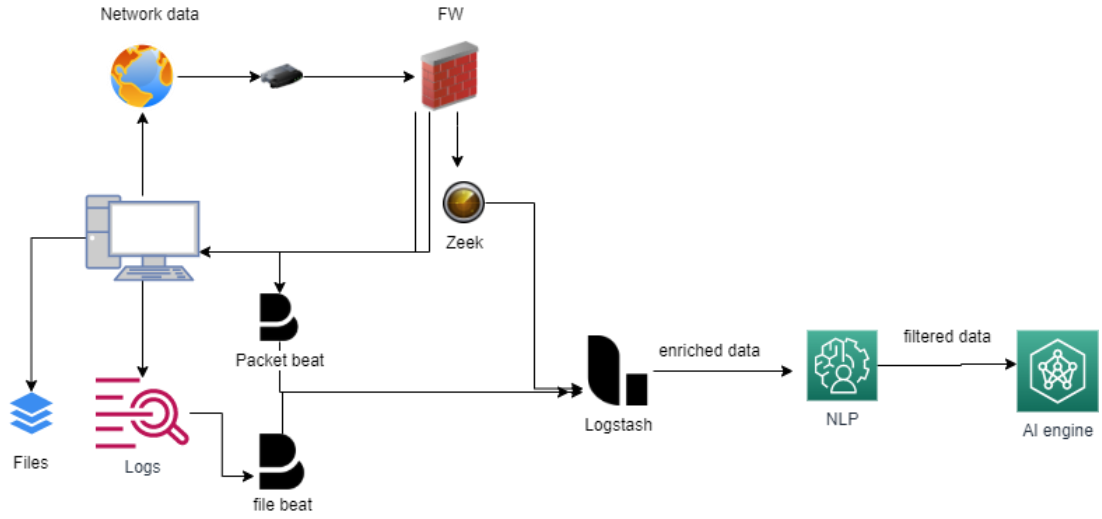


Figure 1Figure 3.1.2.1 Data collecting and processing architecture

3.1.2.1 Capture real time traffic and data from multiple sources

In order to capture data, we use a mechanism called beats to capture data. Beats are lightweight in nature and data collecting varies from each beat, log files are being collected by filebeat, system and services by metric beats and so on [8].

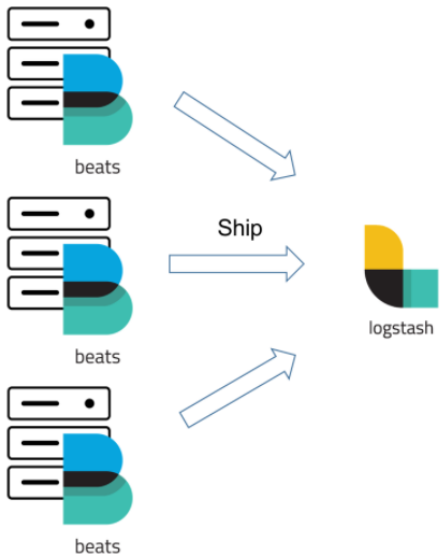


Figure 2Figure 3.1.2.1.1 simple beats architecture [9]

Beats can be used to capture both network and host-based data from the installed client. In order to capture host-based data, it is possible to use filebeat, metricbeat, audit beat and winlogbeat. And use these beats data such as logs, files, programs, processes, and other host-based contents which could be a threat to the system is captured to analyze the legitimacy of it.

In this research we are using filebeat which collects the network logs and passes it to logstash. Using filebeat we can collect the logs which is stored in the system real time. In order to use the filebeat we could specify the path which we need to capture logs from and do other configurations in the filbeat.yml file.

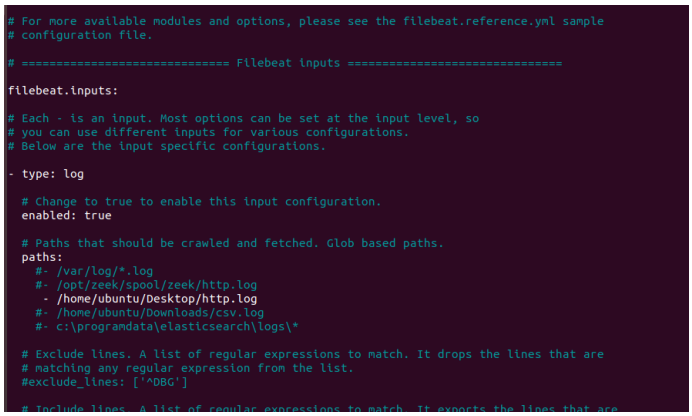


Figure 3Figure 3.1.2.1.2 specifying the path of the logs to be collected.

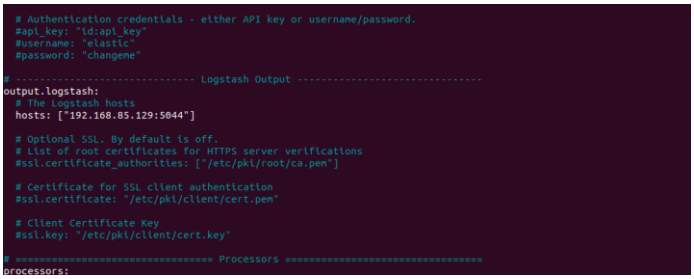


Figure 4Figure 3.1.2.1.3 configuring the output to logstash

Moreover, to capture network traffic we are using packet beats which is another type of beat which captured real time network traffic and we could determine all content accessed in the web and network level such as URL's, hyperlinks, IP addresses etc. Beats could capture large amount of traffic without dropping any data. To configure all the ports and to specify the output paths we need to configure the packetbeat.yml file.

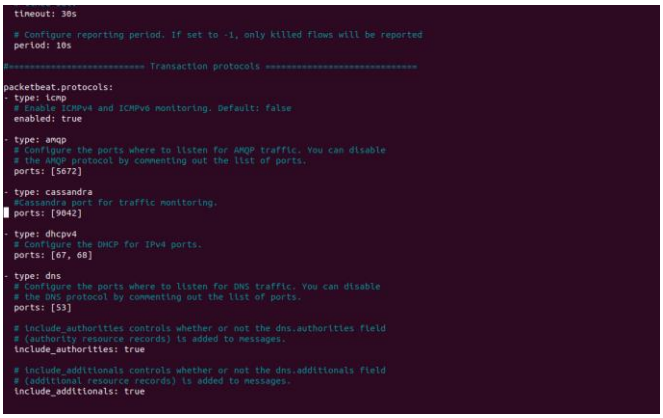


Figure 5Figure 3.1.2.1.5 ports which the traffic is captured from.

in the web and network level such as URL's, hyperlinks, IP addresses etc. Beats could capture large amount of traffic without dropping any data. To configure all the ports and to specify the output paths we need to configure the packetbeat.yml file.

Using the above methods, we are able to capture host-based data using filebeat and network-based data using packetbeat.

Moreover, we have used zeek for more clarity in network-based capture. Zeek is another network capturing tool which we have used in this researched. Using zeek we can capture the network data in a more enriched way.

Once the zeek has been installed in the system it can be used to capture the network traffic and stored in log files real times. From there we could use filebeat to capture the real time logs and pass it to the next step.

Once the zeek server has been started the logs will be stored in the following folder, */opt/zeek/logs/current/*.log.* we could specify the filebeat to collect needed logs from this folder for network monitoring. Once the logs are collected it will be in a format which is not json, therefore which is an issue while integrating with elasticsearch. For the above issue, an addition has been made in a configuration file to convert all logs in json format.

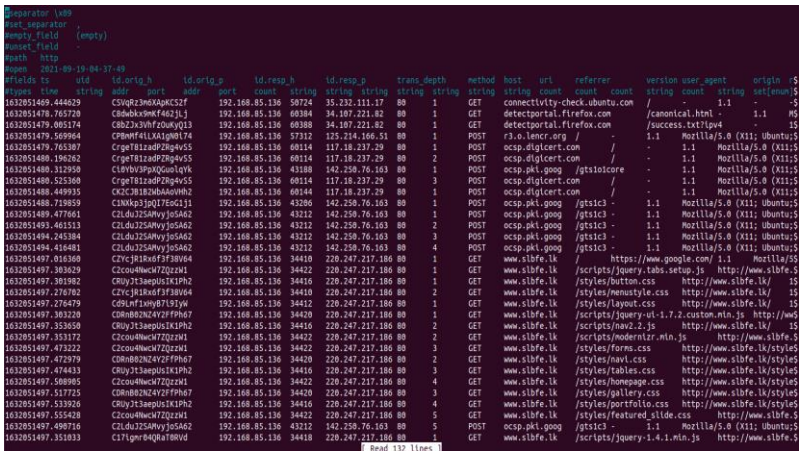


Figure 6Figure 3.1.2.1.10 logs capture before configuring to json

Since the output received is not in json format, we have to convert the it using the following configuration.

```
# Uncomment this to source zkg's package state
# @load packages
#
# Output to JSON
@load policy/tuning/json-logs.zeek
```

Figure 7Figure 3.1.2.1.11 convert the logs to json

Once the configuration has been made the logs file outputs will be in json format so that it is readable by elasticsearch and does all other monitoring.

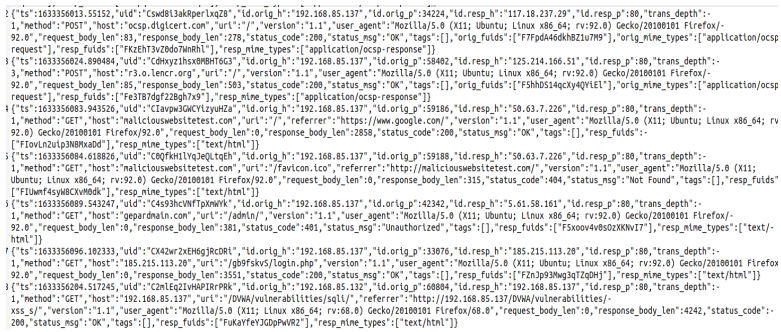


Figure 8Figure 3.1.2.1.12 zeek logs in json format

Once all the setup has been done to capture network traffic, it will be passed for logstash to do the filtering.

3.1.2.2 Process the captured data into a meaningful format

This way we could capture relevant real time data and then we forward the captured data to logstash which is a mechanism used to enrich the received data, using logstash we could process the data and enrich it, and since the data is enriched, it is clearer and more formatted and it increases the accuracy level for decision making in detecting threats. Using logstash we could process data collected from anywhere and use it for further analysis and since the output of the captured packet looks unclear, we have to process the data in the packet and enrich the data to find anomalies in the system [8].

Using logstash, the received data can be transformed into any desired format and forward it to a desired place where the results could be displayed. In this research we will be using elasticsearch to receive the data outputted from logstash and use kibana to visualize it.

The logstash file extention is meant to be a .conf file where it contains three main parts as input, filter and output. These 3 parts can do significant job using data. Input field can receive data from many sources, whereas filter field has many filtering techniques and those filtered data can be outputted to many desired sources.

Figure 9Figure 3.1.2.2.1 simple logstash file

The above figure shows the input from tcp capture whereas our input is from beats and the conf file changes according to it.

The filter section in logstash is used to filter the captured data according to expected output and in a meaningful format. Logstash has many filtering techniques that can be used to filter data, and process the data to the output source. There are filters such as

grok patterns, kv pairs, json, metrics, mutate and so on. In this research we will be using grok patterns and kv pairs in order to filter out the data according to the expected output.

Figure 10Figure 3.1.2.2.2 logstash filter with kv pairs

Figure 3.1.3.2 shows the logstash filtering using kv pairs. The action done in the above kv pair is to split all the fields in the log files separately so that the AI engine and the Neuro realistic engine could get the captured data by specifying the relevant field.

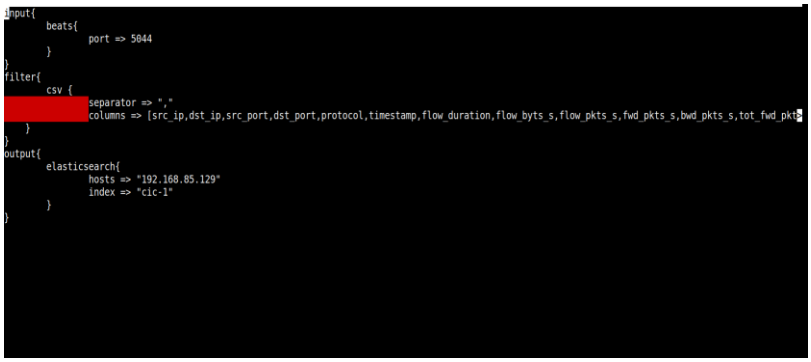


Figure 3.1.2.2.3 shows the logstash filter csv where the logstash files receives the input which is in a csv format and then it separates the fields to its column name and passes it to elasticsearch for the AI engine to get the data and input to the python script to conduct the threat detection.

Once the filtering part has been done, next it moves to output sections of the logstash conf file. Logstash can output to many sources in many formats. Some of the methods are csv, file, elasticsearch, file etc.

In our research the output will be passed to elasticsearch and from there onwards the AI engine and the Neuro realistic detection will collect those data for further analysis.

In order to send the logs to elasticsearch, we have to create an index and all the data will be stored inside that index. The index can be viewed in the discover tab in kibana dashboard.

In order to send the logs using logstash, firstly we have to run the logstash server using the below command. In order to run the command, we have to be in the /usr/share/logstash directory.

. /bin/logstash --path.settings=/etc/logstash/ -f /etc/logstash/conf.d/zeek.conf

Once the logstash server has been started, using the beats all the data will be captured real time and sent to elasricsearch.

Using the above index, we could visualize all the data and needed information, but when it comes to the autonomous cyber AI these fields in the index will be extracted and put into a python script where it is involved with AI models so that it could detect if these data are malicious or not, once the analysis has been done, it will be passed back to elasticsearch using an index and we could use the kibana dashboard to visualize these data.

3.1.3 Functioning and Implementation of AI engine

3.1.3.1 Getting started with building AI engine

After having several detailed discussions with team members, supervisor and co supervisor all the necessary information and features of the proposed AI engine were identified properly. In addition to that core concepts of the intrusion detection arena under the cyber security domain were carefully observed and the strengths and the drawbacks of the existing products were discovered in while doing the literature survey. So, under this chapter it is explained all the steps were taken in detail while building the proposed AI engine.

3.1.3.2 Data gathering phase

Algorithms that use in both machine learning and deep learning look for certain patterns and predict future changes based on them. So it is crucial to select proper data collection techniques when building predictive models since they are only as good as the data they were trained on. Due to this reason to start with building the proposed AI engine, the first thing needed to do was gather most suitable datasets. (Sets of data gathered and stored in a certain manner are referred to as dataset)

3.1.3.3 Description of Datasets

As previously discussed, IDS are widely used as the second line of defense in almost all the computer networks out there in the world. Even though the topic intrusion detection using machine learning and deep learning is not a new topic to the cyber security domain during the literature survey we were able to identify that almost all the research that were done previously utilized KDD99, NSL-KDD dataset. Therefore in this research NSL-KDD dataset also used to evaluate the performance of the proposed model to compare the performance with related work since it was widely utilized as a benchmarking dataset for network intrusion detection systems and considered as an improved version which is free of defects that were there with KDD99 dataset.

But ultimate goal of this research is to provision customers a reliable and user-friendly product so for that the accuracy and the reliability of the machine learning and deep learning models occupied in the product is highly important. Hence, in order to achieve the higher accurate and consistent performance it is a must to use a reliable dataset for training and testing purposes. In this research CICIDS is used as the benchmarking dataset which is already known as most up to date reliable dataset and it was created by the Canadian Institute of Cyber Security. The CICFlowMeter was used to monitor and capture the network traffic in a predefined lab environment, in the lab environment they used 4 computers with 1 server and 1 switch as attack infrastructure and 10 computers with 3 servers and 2 switches as the victim infrastructure and the flowmeter tool ended up giving a label to each record as well as providing information about the captured traffic's originated source IP address and destination IP address and port numbers, as well as the timestamp related information about each record. Several protocols were occupied like HTTPS, HTTP, SSH, SMTP while simulating the attacks in the testing environment.

3.1.3.4 Data cleaning

When it comes to using machine learning programmers often have difficulties supplying the correct data to ML and DL algorithms by clearing out unnecessary and error-prone data and spent considerable amount of time to clean datasets to generate an error-free dataset. In this research also it was carefully observed and removed all the missing values, unexpected noises, values with unstructured formats, redundancy issues of the records and columns of both NSL KDD and CICIDS datasets before using them. Because if we feed those data to the machine learning and deep learning models it could lead to decrease the accuracy and efficiency of those models.

3.1.3.4.1 More about CICIDS

When it comes to the CICIDS dataset it was comprised with the IP addresses of the originating source's and the destination host’s in the original network, it was essential to remove particular data in order to obtain unbiased detection, since utilizing

such information while training may could cause to overfit the trained model. After that process it was ensured that all the ML and DL models would extract the features and learn from the packets regardless to their source and destination information. Apart from that some attack categories contained weird characters and they also renamed to refrain from misclassifying issues.

```
[ ]  dataset['Label'].value_counts()

benign          2271320
hulk             230124
portscan         158804
ddos             128025
goldeneye         10293
ftppatator         7935
sshpatator         5897
slowloris          5796
slowhttptest       5499
bot                1956
bruteforce         1507
xss                 652
infiltration         36
sql                  21
heartbleed           11
```

Figure 11Figure 3.1.3.4.1.1: Attack distribution of the original dataset

After the cleaning process it is observed that there was lack of attack instances of Sql, Infiltration and Heartbleed categories and comparatively high number of benign instances. Therefore those instances were dropped and using CICFlowmeter we generated our own set of Sql attack instances and new variant called Synflood which falls under denial-of-service attack category and added them back to the original dataset with fraction of Benign traffic instances obtained from the original dataset. And following graph shows the modified datasets attack distribution.

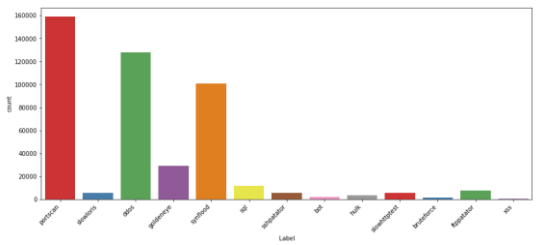


Figure 12Figure 3.1.3.4.1.2: Attack distribution of the modified dataset

3.1.3.4.2 Dealing with imbalance nature of the dataset

When building ML and DL models for real world scenarios, it is common issue to run across class imbalances in the datasets that are being used to train models. This happens when one category has a much greater number of instances than others. As you can see in this research, we also had to encounter with that problem when it comes to the CICIDS dataset. For an instance in our dataset the amount of port scan, ddos and synflood attack instances were relatively higher than other attack categories. To overcome this problem several techniques were used, and they are listed as follows,

- Upsampling - This is process of generating artificial data points which corresponds to the minor attack categories of the dataset and injecting back into the original dataset. After the execution of this process all the instances of attack categories of the datasets become relatively identical to each other. This equalizing mechanism terminates the ML and DL models being adapting to one or two dominant classes in the dataset Furthermore, the extra information added in the upsampling process adds bias into the system in a positive manner.

- Downsampling – This process is also similar as previously explained upsamling but as the name explains it works in the opposite way. In this case classes that corresponds to the majority instances were reduced to the level of minor class instances, but this caused to loss of important information and this method did not work in our case as expected.

- Assigning weights - In this process different weight values are allocated to the different classes in the dataset. When compared attack instances of the majority classes, the attack instances with minority classes will be granted higher weight values after completion of this process. Therefore, the model will treat all classes equally in the training process. In our case models trained with this technique showed better improvement than in the previous methods.

3.1.3.5 Dealing with categorical data of the datasets

When it comes to machine learning there are two main types of data available in the majority of datasets. They are differentiated as numerical and categorical data. As the name explains numerical data comprise with different types of numbers such as integer values float values and categorical values comprise with textual data. Since machine leaning and deep learning models are mathematical in their nature most of them only deal with numbers and that is one of the major reasons to convert those textual data into numerical representation before feeding them into machine learning and deep learning models.

When it comes to NSL-KDD dataset it contained 43 features including 4 categorical features and they were protocol type, service flag and attack label. Here we encoded all 3 categorical features except the attack label and then they were converted to numerical values by using one hot encoding approach.

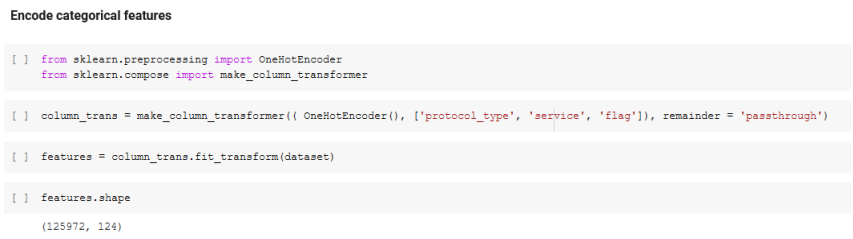


Figure 13Figure 3.1.3.5.1: Encoding categorical features of NSL-KDD dataset

Then the remaining attack label column was separated and encoded that using label encoding approach as shown in the image below.

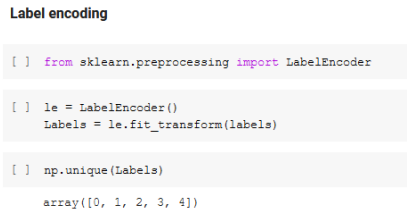


Figure 14Figure 3.1.3.5.2: Encoding process of the attack labels of NSL-KDD dataset

Even though there were 78 features contained in the CICIDS dataset after data wrangling process there was only one column with textual data, and it was attack label column. So here also label encoding was used as the encoding approach as shown in the image below.

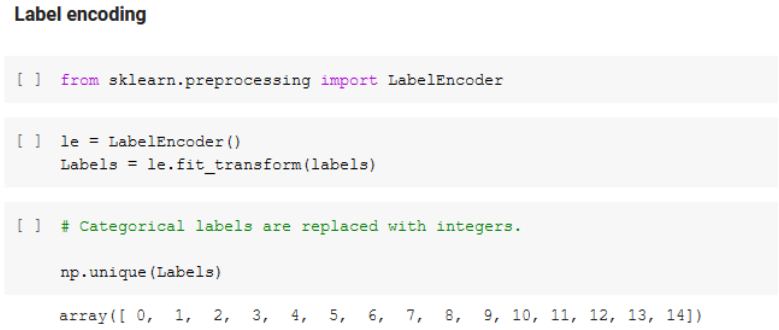


Figure 15Figure 3.1.3.5.3: Encoding process of the attack labels of CICIDS dataset

After the encoding process all the attack names in both datasets converted into integers.

3.1.3.6 Description of Machine learning and Deep learning models.

In this section we initially present the theoretical concepts behind used deep learning and machine learning models utilized throughout this research. Furthermore both unsupervised and supervised approaches taken into consider selecting the best model before building the AI engine and all the experiments and corresponding results are explained in detail under the result and discussion section.

3.1.3.6.1 Support Vector Machine

Support Vector Machine is a machine learning algorithm which use supervised approach for solving problems related with data classification and regression and work well for many real-world anomaly detection scenarios. The basic idea behind SVM is segregating the given dataset into different classes depending on the features. For that it uses a hyperplane which is considered as a decision plane or decision boundary line. In here the number of features in the given dataset and the

dimension of the hyperplane are correlated with each other. For an instance consider this hyperplane as a two-dimensional line that connects all of the given input data points. When the SVM learning technique is used, the optimum hyperplane separation between classes is determined by the coefficients. The margin measures the distance between the hyperplane and the closest data points. The line with the largest margin is the optimal or ideal hyperplane for dividing the two classes. These are the only variables that are taken into consideration while designing the hyperplane and classifier. The technical name for these data points is called as support vectors.

3.1.3.6.2 Decision Tree

Just like previously explained support vector machine algorithm, decision tree also uses supervised learning-based approach solving problems related with data classification and regression. But in real world scenarios it is widely used for classification problems. As the name explains this algorithm has a tree like architecture where features of the dataset are represented by the internal nodes and decision rules of the algorithm are represented by branches. When it comes to nodes, one type was used to make decisions and one was used to represent taken decisions. They are called as decision node and leaf node respectively. When it comes to internal process it starts from the node call root node which represents the whole dataset and begin to find the most suitable attributes of the supplied dataset. Then algorithm divides it into subsets based on the bets attribute values. Then decision node is generated which comprise with the best attribute and continues generating decision trees by repeating those steps until it fails to classify nodes further.

3.1.3.6.3 Naïve Bayes

Naïve bayes algorithm also uses supervised learning-based approach solving problems related with data classification and it is based on bayes theorem, so it

usually makes decisions by assuming there is not any correlation between the features with each other for any given dataset for training purpose. Moreover there are three types of naïve bayes algorithms known as Gaussian, Multinomial and Bernoulli and in this research Multinomial model was used.

3.1.3.6.4 Random Forest

This is also based on supervised learning, and it uses a technique called as ensemble learning which combines predictions from multiple decision trees that run parallelly without interacting with each other to provide more accurate and reliable prediction. Basically what happens is algorithm itself calculate the mean value of the classes as the prediction of all the decision trees and gives the output as the final output.

3.1.3.6.5 Autoencoders

Autoencoders are unsupervised deep learning algorithm which takes input values and send it through hidden layers and reconstruct the output data which are identical to the original input data moreover, it utilizes a compression and decompression function throughout that process and following are the major components of a general autoencoder, which contribute to the learning process of autoencoders.

Encoder: This part is responsible for compressing the input data in other words, the dimension reduction happens in this part, and this helps to figure out the prominent hidden features of the input dataset.

Bottleneck: This part is made up with the lowest number of hidden nodes and comprises the compressed or encoded new set of features which represents the original input data.

Decoder: This part of the autoencoder is responsible for reconstructing the output values from the compressed data which are highly equivalent to input dataset.

These autoencoders are data specific, which turns out the meaning that these are only able to compress or encode data which are identical or have close features to what they have been previously trained on. Hence it can be utilized to distinguish the difference between normal network traffic and anomalous network traffic with high accuracy.

3.1.3.6.6 Artificial Neural Network

Approximately the human brain is made up hundred billion neurons in which each one is connected to thousands of neighbor neurons which basically gives the ability to learn, adapt and apply based on previous experience. The main idea behind artificial neural network is that to provide the potential ability to machines to mimic how the human brain operates where the artificial structure is created called as artificial neural network that comprises of the input layer of neurons, one or more hidden layers of neurons and an output layer moreover, input values are processed through all the interconnected hidden neural network layers just like the human brain and finally gives the output value. These are also known as feed forward neural networks as inputs are processed only in the forward direction.

In this research it is tested FFNN which is an ANN for both NSL-KDD and CICIDS datasets and results proved that this approach outperformed than all the other deep learning and machine learning approaches. Hence this used to build the AI engine of the Autonomous Cyber AI product. To build this model in robust manner several techniques were used, one was using L2 regularization to prevent the model from over fitting and other method was to adding drop out layer which disables some hidden neurons in the training process.

3.1.3.7 Performance Matrices

Following metrics will be utilized to evaluate the performance of the proposed deep learning models and the machine learning models throughout this research project [18].

TP (True Positive): The data instances correctly predicted as attacks by the classifier.

FN (False Negative): The attack data instances incorrectly predicted as normal by the classifier.

FP (False Positive): The normal data instances incorrectly predicted as attacks by the classifier.

TN (True Negative): The data instances correctly predicted as normal by the classifier

Accuracy: = TP + TN / TP + TN + FP + FN.

Precision: = TP / TP + FP.

Recall: = TP / TP + FN.

F1 Score: = 2*(Recall * Precision) / (Recall + Precision)

3.1.4 Functioning and Implementation of Neuro Realistic detection engine.

3.1.4.1 System architecture

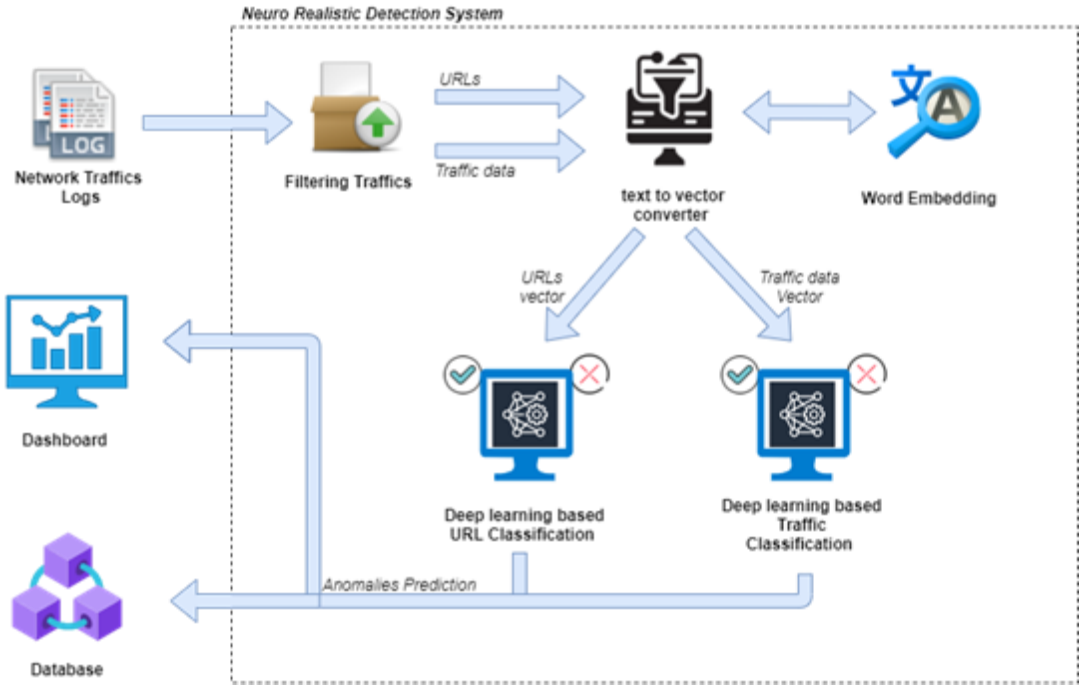


Figure 16Figure 3.1.4.1.1 High Level Architectural Diagram

My research main objective is to detect anomalies network traffics using NLP based classification. For that as shown in Figure 4.1.1, firstly network logs are taken as the input to the system. For increase the efficiency of the training filter out the related data while normalizing the data. After extract URLs and necessary http traffic's data separately and pass them to the word embedding algorithm. It generates a list of lists formatted custom corpus since word2vec model requires a corpus as the input. In the training word2vec model converts the input data into vector while tokenizing the words. Next, the preprocessed vectors are input to respected classification algorithms. In the proposed system there are two different classification algorithms to classified malicious URLs and anomalies http traffics. The classified outputs are visualized on a user dashboard and stored in a blockchain based database. Furthermore, backend of the system will be running on the cloud to overcome the lack of performance in end

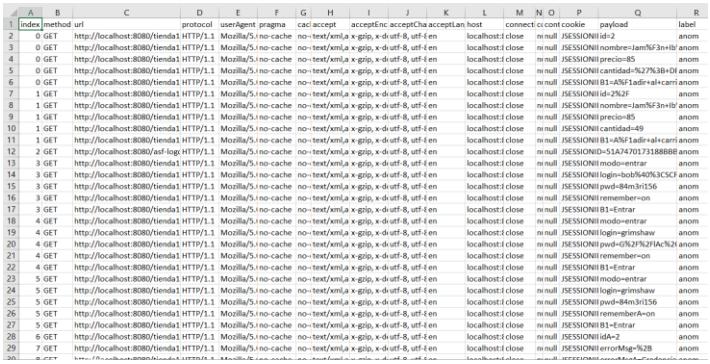
user systems. The ultimate objective is improving the system to detecting anomalies network traffics in real-time accurately.

3.1.4.2 Datasets

As first step, I search about datasets which are related to network traffics. To implement and test my deep learning models, datasets are needed. While researching I found datasets called KDDcup99 [11], UNSW-NB15 [12] and HTTP DATASET CSIC 2010 [13]. KDDcup99 dataset was implemented in 1999 based on network traffics and it contained 22 types of network attacks under four categories as shown on Table 4.1.1.

In UNSW-NB15 dataset contains two million and 540,044 data instance which are stored in four CSV files. A partition from those datasets were configured for training and testing by including 175,341 instances for training and 82,332 instances for testing. But when considering the features of those dataset, they are contained with only non-textual data like shown in Figure 4.1.2 and Figure 4.1.2. Though those datasets fit for machine learning models, they are not fulfilling the requirements of training NLP based model. By training with those datasets cause to increase the complexity of the model and decrease the accuracy level of the model. Therefore, I decided to not take those datasets.

Next, I considered the HTTP Dataset CSIC 2010 dataset which were developed by "Information Security Institute". It contains textual data which captured from upper layer of OSI model including URLs, traffic methods, cookie values and payloads. The web traffics were generated automatically by mainly focusing web attacks and labeled as normal or anomalous. Moreover, the dataset includes attacks such as SQL injection, buffer overflow, information gathering, files disclosure, CRLF injection, XSS, server-side include, parameter tampering. Figure 4.1.4 shows the content of the HTTP CSIS dataset. Since this dataset compatible with NLP algorithms, I decided to select HTTP Dataset CSIS 2010 dataset to train my deep learning model.



index	method	url	protocol	userAgent	pragma	cac	accept	acceptEnc	acceptCha	acceptLan	host	connecti	ci	cont	cookie	payload	label
0	GET	http://localhost:8080/tienda1	HTTP/1.1	Mozilla/5.(	no-cache		no-text/xml,a	x-gzip, x-d	utf-8, utf-f	en	localhost:(	close	ni	null	JSESSIONII	id=2	anom
0	GET	http://localhost:8080/tienda1	HTTP/1.1	Mozilla/5.(	no-cache		no-text/xml,a	x-gzip, x-d	utf-8, utf-f	en	localhost:(	close	ni	null	JSESSIONII	nombre=Jam%F3n+Ib	anom
0	GET	http://localhost:8080/tienda1	HTTP/1.1	Mozilla/5.(	no-cache		no-text/xml,a	x-gzip, x-d	utf-8, utf-f	en	localhost:(	close	ni	null	JSESSIONII	precio=85	anom
0	GET	http://localhost:8080/tienda1	HTTP/1.1	Mozilla/5.(	no-cache		no-text/xml,a	x-gzip, x-d	utf-8, utf-f	en	localhost:(	close	ni	null	JSESSIONII	cantidad=%27%3B+DI	anom
0	GET	http://localhost:8080/tienda1	HTTP/1.1	Mozilla/5.(	no-cache		no-text/xml,a	x-gzip, x-d	utf-8, utf-f	en	localhost:(	close	ni	null	JSESSIONII	B1=A%F1adir+al+carri	anom
1	GET	http://localhost:8080/tienda1	HTTP/1.1	Mozilla/5.(	no-cache		no-text/xml,a	x-gzip, x-d	utf-8, utf-f	en	localhost:(	close	ni	null	JSESSIONII	id=2%2F	anom
1	GET	http://localhost:8080/tienda1	HTTP/1.1	Mozilla/5.(	no-cache		no-text/xml,a	x-gzip, x-d	utf-8, utf-f	en	localhost:(	close	ni	null	JSESSIONII	nombre=Jam%F3n+Ib	anom
1	GET	http://localhost:8080/tienda1	HTTP/1.1	Mozilla/5.(	no-cache		no-text/xml,a	x-gzip, x-d	utf-8, utf-f	en	localhost:(	close	ni	null	JSESSIONII	precio=85	anom
1	GET	http://localhost:8080/tienda1	HTTP/1.1	Mozilla/5.(	no-cache		no-text/xml,a	x-gzip, x-d	utf-8, utf-f	en	localhost:(	close	ni	null	JSESSIONII	cantidad=49	anom
1	GET	http://localhost:8080/tienda1	HTTP/1.1	Mozilla/5.(	no-cache		no-text/xml,a	x-gzip, x-d	utf-8, utf-f	en	localhost:(	close	ni	null	JSESSIONII	B1=A%F1adir+al+carri	anom
2	GET	http://localhost:8080/asf-logi	HTTP/1.1	Mozilla/5.(	no-cache		no-text/xml,a	x-gzip, x-d	utf-8, utf-f	en	localhost:(	close	ni	null	JSESSIONID	=51A747017318BBE	anom
3	GET	http://localhost:8080/tienda1	HTTP/1.1	Mozilla/5.(	no-cache		no-text/xml,a	x-gzip, x-d	utf-8, utf-f	en	localhost:(	close	ni	null	JSESSIONII	modo=entrar	anom
3	GET	http://localhost:8080/tienda1	HTTP/1.1	Mozilla/5.(	no-cache		no-text/xml,a	x-gzip, x-d	utf-8, utf-f	en	localhost:(	close	ni	null	JSESSIONII	login=bob%40%3CSCF	anom
3	GET	http://localhost:8080/tienda1	HTTP/1.1	Mozilla/5.(	no-cache		no-text/xml,a	x-gzip, x-d	utf-8, utf-f	en	localhost:(	close	ni	null	JSESSIONII	pwd=84m3ri156	anom
3	GET	http://localhost:8080/tienda1	HTTP/1.1	Mozilla/5.(	no-cache		no-text/xml,a	x-gzip, x-d	utf-8, utf-f	en	localhost:(	close	ni	null	JSESSIONII	remember=on	anom
3	GET	http://localhost:8080/tienda1	HTTP/1.1	Mozilla/5.(	no-cache		no-text/xml,a	x-gzip, x-d	utf-8, utf-f	en	localhost:(	close	ni	null	JSESSIONII	B1=Entrar	anom
4	GET	http://localhost:8080/tienda1	HTTP/1.1	Mozilla/5.(	no-cache		no-text/xml,a	x-gzip, x-d	utf-8, utf-f	en	localhost:(	close	ni	null	JSESSIONII	modo=entrar	anom
4	GET	http://localhost:8080/tienda1	HTTP/1.1	Mozilla/5.(	no-cache		no-text/xml,a	x-gzip, x-d	utf-8, utf-f	en	localhost:(	close	ni	null	JSESSIONII	login=grimshaw	anom
4	GET	http://localhost:8080/tienda1	HTTP/1.1	Mozilla/5.(	no-cache		no-text/xml,a	x-gzip, x-d	utf-8, utf-f	en	localhost:(	close	ni	null	JSESSIONII	pwd=0%2F%2FIAc%2(	anom
4	GET	http://localhost:8080/tienda1	HTTP/1.1	Mozilla/5.(	no-cache		no-text/xml,a	x-gzip, x-d	utf-8, utf-f	en	localhost:(	close	ni	null	JSESSIONII	remember=on	anom
4	GET	http://localhost:8080/tienda1	HTTP/1.1	Mozilla/5.(	no-cache		no-text/xml,a	x-gzip, x-d	utf-8, utf-f	en	localhost:(	close	ni	null	JSESSIONII	B1=Entrar	anom
5	GET	http://localhost:8080/tienda1	HTTP/1.1	Mozilla/5.(	no-cache		no-text/xml,a	x-gzip, x-d	utf-8, utf-f	en	localhost:(	close	ni	null	JSESSIONII	modo=entrar	anom
5	GET	http://localhost:8080/tienda1	HTTP/1.1	Mozilla/5.(	no-cache		no-text/xml,a	x-gzip, x-d	utf-8, utf-f	en	localhost:(	close	ni	null	JSESSIONII	login=grimshaw	anom
5	GET	http://localhost:8080/tienda1	HTTP/1.1	Mozilla/5.(	no-cache		no-text/xml,a	x-gzip, x-d	utf-8, utf-f	en	localhost:(	close	ni	null	JSESSIONII	pwd=84m3ri156	anom
5	GET	http://localhost:8080/tienda1	HTTP/1.1	Mozilla/5.(	no-cache		no-text/xml,a	x-gzip, x-d	utf-8, utf-f	en	localhost:(	close	ni	null	JSESSIONII	rememberA=on	anom
5	GET	http://localhost:8080/tienda1	HTTP/1.1	Mozilla/5.(	no-cache		no-text/xml,a	x-gzip, x-d	utf-8, utf-f	en	localhost:(	close	ni	null	JSESSIONII	B1=Entrar	anom
6	GET	http://localhost:8080/tienda1	HTTP/1.1	Mozilla/5.(	no-cache		no-text/xml,a	x-gzip, x-d	utf-8, utf-f	en	localhost:(	close	ni	null	JSESSIONII	idA=2	anom
7	GET	http://localhost:8080/tienda1	HTTP/1.1	Mozilla/5.(	no-cache		no-text/xml,a	x-gzip, x-d	utf-8, utf-f	en	localhost:(	close	ni	null	JSESSIONII	errorMsg=%2B	anom

Figure 17Figure 3.1.4.2.1 HTTP Dataset CSIS 2010 dataset

I had to find another dataset to train a model to classify malicious URLs, since the HTTP CSIS dataset were generated in localhost and captured URLs cannot used to train my model. Because of that I moved on to find another dataset which contains malicious URLs, and I found a dataset. It contains both malicious and normal URLs for training and it was created by Faizan Ahmad (Fulbright computer science undergrad and CEO of Fsecurify) [14] and he has used a web crawler for collected malicious URLs out from websites and labeled both malicious and normal URLs where is shown in Figure 4.1.5. The dataset contains around 320,000 normal URLs and 80,000 malicious URLs.

Along with the those selected datasets, me and my research group members decided to create new dataset for train our models. Therefore, our group member Hashmi started to develop a dataset. He was able to generate a dataset by capturing network traffic using Filebeat along with ELK stack [22] and manually labeled. It contained with host URL, IP, user agent, method, request and response body length, status message as shown in figure 4.1.6.

	A	B
1	url	label
2	diaryofagameaddict.com	bad
3	espdesign.com.au	bad
4	iamagameaddict.com	bad
5	kalantzis.net	bad
6	slightlyoffcenter.net	bad
7	toddscarwash.com	bad
8	tubemoviez.com	bad
9	ipl.hk	bad
10	crackspider.us/toolbar/install.php?pack=exe	bad
11	pos-kupang.com/	bad
12	rupor.info	bad
13	svision-online.de/mgfi/administrator/components/com_babackup/classes/fx29id1.txt	bad
14	officeon.ch.ma/office.js?google_ad_format=728x90_as	bad
15	sn-gzzx.com	bad
16	sunlux.net/company/about.html	bad
17	outporn.com	bad
18	timothycopus.aimoo.com	bad
19	xindalawyer.com	bad
20	freeserials.spb.ru/key/68703.htm	bad
21	deletespyware-adware.com	bad
22	orbowlada.strefa.pl/text396.htm	bad
23	ruiyangcn.com	bad
24	zkic.com	bad
25	adserving.favorit-network.com/eas?camp=19320;cre=mu&grpid=1738&tag_id=618&nums=FGApl	bad

Figure 18Figure 3.1.4.2.2 : Malicious URLs dataset

	A	B	C	D	E	F	G	H	I
1		method	id.resp_p	version	host	url	user_agent	status_msg	response_body_len
2	0	GET	80	1.1	www.galadarlhotel.lk	/images/home/home	Mozilla/5.0 (X11; Ub	Not Modified	0
3	1	GET	80	1.1	www.galadarlhotel.lk	/images/home/home	Mozilla/5.0 (X11; Ub	Not Modified	0
4	2	GET	80	1.1	www.forestdept.gov.l	/images/forest/home/	Mozilla/5.0 (X11; Ub	OK	47176
5	3	GET	80	1.1	platform.twitter.com	/widgets.js	Mozilla/5.0 (X11; Ub	OK	97806
6	4	GET	80	1.1	i1.cdn-image.com	/__media__/js/min.js	Mozilla/5.0 (X11; Ub	OK	8435
7	5	GET	80	1.1	www.dmc.gov.lk	/modules/mod_sp_ni	Mozilla/5.0 (X11; Ub	Not Modified	0
8	6	GET	80	1.1	www.dmc.gov.lk	/images/banners/dm	Mozilla/5.0 (X11; Ub	OK	3841620
9	7	GET	80	1.1	www.slbfe.lk	/styles/homepage.cs	Mozilla/5.0 (X11; Ub	Not Modified	0
10	8	GET	80	1.1	www.naita.gov.lk	/modules/mod_fhw4	Mozilla/5.0 (X11; Ub	OK	2844
11	9	GET	80	1.1	www.sbd.lk	/modules/mod_ppc_	Mozilla/5.0 (X11; Ub	OK	249
12	10	GET	80	1.1	iti.lk	/images/iti/home/pra	Mozilla/5.0 (X11; Ub	Not Modified	0
13	11	GET	80	1.1	www.pgis.pdn.ac.lk	/events/2020/ev2_ici	Mozilla/5.0 (X11; Ub	Not Modified	0
14	12	GET	80	1.1	www.health.gov.lk	/	Mozilla/5.0 (X11; Ub	Not Modified	0
15	13	GET	80	1.1	www.nif.ac.lk	/templates/flex/js/ma	Mozilla/5.0 (X11; Ub	Not Modified	0
16	14	GET	80	1.1	maps.google.com	/maps/vt?pb=!1m5!1	Mozilla/5.0 (X11; Ub	OK	11091
17	15	GET	80	1.1	www.edupub.gov.lk	/images/newsandeve	Mozilla/5.0 (X11; Ub	OK	24514
18	16	GET	80	1.1	www.airforce.lk	/fonts/fontawesome-v	Mozilla/5.0 (X11; Ub	Not Found	229
19	17	GET	80	1.1	floatlanka.gov.lk	/images/home_slide/	Mozilla/5.0 (X11; Ub	OK	25800

Figure 19Figure 3.1.4.2.3 : Network traffic Dataset developed by our team

3.1.4.3 Deep learning model

After selecting proper network traffic-based dataset, my next step was train models for anomaly detection. Since I had to work with NLP techniques, I decided to train deep learning models. Deep learning [15] is a subset of Machine Learning, that attempts to imitates the human brain using neural networks. Deep neural networks contain multiple layers including input layer, hidden layers, and output layer with interconnected nodes as shown in figure 4.1.7. Each node building upon the previous layer to refine and optimize the prediction or categorization. The neural networks have several different forms, including Convolutional Neural Network, Artificial Neural Networks, Recurrent Neural Network, Feed Forward Neural Network. Each network specified for specific use cases. In my case I used Convolutional Neural Network since it fit with NLP.

Since I selected two datasets, I decided to train two separate classification models. One model for classify anomalies network traffics and one model for classify

malicious URLs. Both of selected models were labeled either normal or abnormal, due to that reason I selected binary classification algorithm. It compares each of the predicted probabilities to actual class output which can be either 0 or 1. After the 0,1-classification decoded to normal and abnormal labels.

3.1.4.4 Training Environment

Before starting implementation, I had to choose a platform to training and testing my works. When training deep learning models, it consumes higher CPU and GPU power. Since my local machine's specs are low, I looked for an online platform to train and test my models. After trained I can save the models and reuse them for integration. When searching I found an online platform called "Azure Machine Learning Studio" [17] which hosted by the Microsoft. In azure machine learning studio contain different types of algorithms which related to different kind of areas, so I tried to find way to implement neural network in azure ML studio, but it was not that much possible. There for I searched about another online platform to run deep learning process without any errors from lack of resources. Then I found "Google Colaboratory" [18] which implemented by google to make easy to train machine learning models also google colaboratory compatible with python language. Google colab is online platform as azure machine learning studio but most like Jupiter notebooks because google colab also wants to open separate notebook for each project and implement python code as different sectors to run code. Colab provides GPU TPU processing memories and the cloud spaces. TPU is referred as Tensor Processing Unit. It is an AI accelerator application-specific integrated circuit which developed by Google. The only thing google colab need is a good internet connection and this online platform works properly.

3.1.4.5 Word embedding model

Before implement classification models, I had to select and train word embedding model, because according to findings my research work was use NLP for identify network attacks' features and based on that classify the anomalies network traffics. NLP commonly use training on text data. NLP [19] algorithm take text data as input

and extracts the features from them and tokenize them. Later on, those features can be used with classification model to data classifications. To extract features for anomalies traffic detection I selected to use word2vec algorithm. Word2vec algorithm [20] is a Natural Language Processing algorithm which developed by Google. This algorithm takes word corpus as the input and convert them to a vector representation while tokenizing words by analyzing semantically and syntactically similarity. There are two methods to convert text data into vectors in word2vec naming as continuous-bag-of-words (CBOW) method and skip-gram (SG) method. As shown in figure 4.1.9 CBOW model [9] predicts the probability of the headword by context. Furthermore explaining, let us consider the vocabulary of the training word vectors as V and N be the dimension of the vectors, then the input to the hidden layer would be a matrix WI of size V ×N, where each row represents a word from the vocabulary. In similar way, hidden layer would connect to output layer having a matrix WO, which would be of size N ×V. The Skip-Gram model which shown in figure 4.1.10, predicts the probability of the context by headword. Skip-Gram works well with small number of training data and its training speed is lower than CBOW. In order to, do tokenizing it required a vocabulary

consisting of unique words. The vocabulary is generating when the word2vec model training.

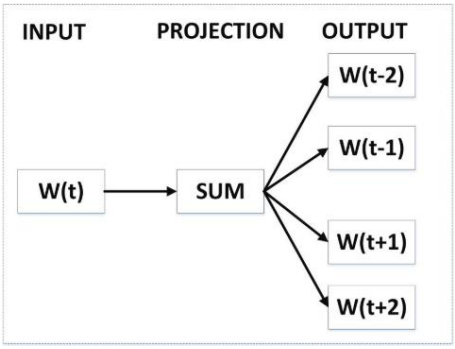


Figure 20Figure 3.1.4.5.1 CBOW Model

As my first step of train word embedding model, I imported the Datasets into Google Colab from the Google Drive and saved it as a data frame. Also imported the word2vec genism model. Before start to generate word vectors I had to pre-process the input data. The inputs are network traffic logs. Since the network traffic logs contain many complex characteristics and unwanted data, they cannot use directly with deep learning

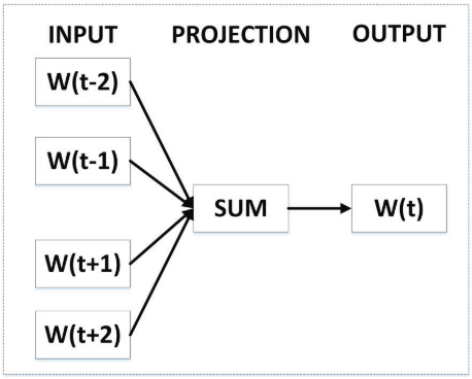


*Figure **3.1.4.5.2**: Skip-Gram Model*

models. It affects to the accuracy and the efficiency of the deep learning algorithms. The data pre-processing cause to reduce the complexity of the input data. The proposed algorithm extracts the relevant data fields and removes the special characters from them and replace them with the spaces while creating a new data frame. Figure 4.1.11 shows the algorithm which used to remove special characters from network traffics to reduce the complexity.

```
DEFINE FUNCTION clean-text (INPUT):
  SET special characters EQUAL ["!",'"',
                "#","%","&","'","(",")",
                "*","+",",","-",".","/",
                ":",";","<",
                ">","?","@","[","\\",
                "]","^","_","`","{","|",
                "}","~","-"]
  FOR characters IN special characters:
      OUTPUT EQUAL INPUT REPLACE(characters TO 'space')
  RETURN OUTPUT
```

Figure 21Figure 3.1.4.5.3: Special Characters replacing algorithm

After pre-processed, I split the data frame for training and testing. For training I have taken 70% and other 30% for testing. Figure 4.1.12 shows the data pre-processing workflow. When training word2vec model, it required mention the parameters. Details about those parameters are described in Table 4.1.2. I trained the two word2vec models using CBOW method and Skip-Gram. For training, I created new data column by combined relevant data columns expect the labels from my dataset. Then it used for

the train the word2vec models. After training, each model vector size was 100. Then, combining those two vectors, I created new embedding matrix which contain 200 size of word vector. Later on, this embedding matrix can be used with deep neural classification models as a sequential layer.

3.1.4.6 Classification model

My next step was the implement the classification models. For that I selected the TensorFlow [21] library. TensorFlow implemented by google as opensource library for large numerical calculations. Keras integrated with TensorFlow enable to implement pipeline basically keras running on TensorFlow. TensorFlow act as large tensors, so NumPy library imported for handle large arrays and matrices. Pandas with NumPy, pandas got ability to deal with perform data analysis and deep learning tasks. Combination of these libraries and the Colab online platform made the best environment for implement my models. In this phase I dealt with build Convolutional

Figure 22Figure 3.1.4.6.1: Relu activation function

Neural Network (CNN) [20] including ‘relu, sigmoid’ function with ‘binary-cross-entropy’. Rectified Linear Unit (Relu) activation function allow DL models to learn faster if output is positive then send output if not it sends zero as output as shown on Figure 4.1.13. It’s a linear function. Sigmoid activation function makes all in between 1 and 0 as “S” shaped as shown on Figure 4.1.14. It’s a non-linear function. Binary cross entropy [16] uses to calculate loss of DL model because it predicts outputs between 1 and 0. Furthermore, minimize the average probability error between the target and predicted label for each embedding.

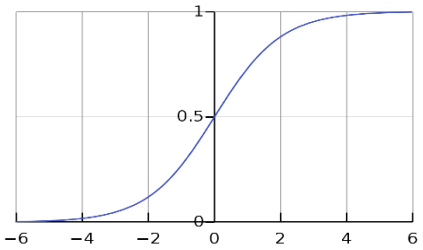


Figure 23Figure 3.1.4.6.2: Sigmoid activation function

Source 1: https://en.wikipedia.org/wiki/Sigmoid_function

The Neural network was build using dense layers, which make connection between neurons. When it came for CNN [20], every layer tries to find a pattern or relationship of input data by sliding kernel/filter window. In text classification the kernel looks for embedding values of the words and add weights according to patterns. To implement my classification model, I choose one-dimension CNN. In 1D CNN, the kernel window only moves one-by-one down in an embedding list like shown in figure 4.1.15. Then multiplied the embedding values by the kernel weights and summing all multiplied values to get an output. After use maxpooling to discard less-relevant features.

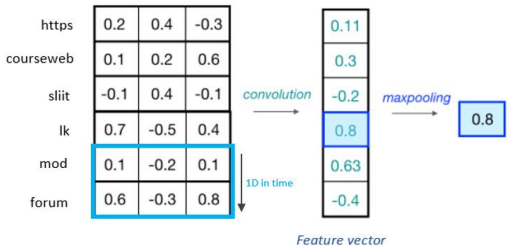


Figure 24Figure 3.1.4.6.3: 1D CNN process

Figure 3.1.4.6.3 describes how neural network implements with 1D CNN layer using python and keras sequential layers, according to this implemented neural network contains an input layer, a reshape layer to reshape the input layer to be compatible with the 1D convolution layer, three 1D convolution layers, a pooling layer for select the maximum element from the region of the feature map covered by the 1D filter, and a flatten layer to reshape the convolution layer output, finally it consists of three dense layers with activation functions. Here it used relu activation function and sigmoid activation function. As discussed earlier, trained word embedding model was taken for the input layer. Then the model compile with adam optimizer for increase efficiently of learning with less memory consumption. Also compile with the Binary Cross entropy. Figure 4.1.17 shows the workflow of the model training. After implementing all these resources, I started the train the classification model using pre-processed training data. Before entering training data, I tokenized those input words using a Tokenizer. The tokenizer converts input data to an array while assigning tokens for each word. To define tokens, tokenizer should be pre-trained using a word corpus. For that I used my train dataset. After tokenizing, all array value length must be same. Therefore, I added pads by defining fixed array size as 160. After all I started the training process. I gave 100 epochs and 128 batch sizes to run with train data as testing purposes. The epoch parameter defines the number of times the training model learn with the entire training dataset and the batch size defines the number of samples processed before update the model. Colab took about 2 hours for complete the train.

After that, I built another classification model to classify malicious URLs as shown on Figure 4.1.18. For this model I have used an input layer with word2vec embedding matrix which created by me, a reshape layer, two 1D convolution layers, a pooling layer, dropouts, also a flatten layer, and three dense layers with relu and sigmoid

activation functions. Then I tested out both models using selected datasets and our dataset. By analyzing results, I realized that the models give good accuracy when train models using combination of selected and our dataset. After considering about all

3.1.4.7 Integrate with ELK stack

My next step was, integrate my models with ELK [23] stack and test out with real time data. ELK is referring three open-source projects which are Elasticsearch, Logstash, Kibana. Elasticsearch is an analytics and search engine. Logstash used to processing captured data in server side and send to elasticsearch stash and the Kibana used to visualize those data using charts and graphs. Here my plan was, retrieve the network traffic logs which are capturing in real time from elasticsearch. Then process those data and classified using my DL models and transfer them to elasticsearch again for visualize with Kibana. Therefore, I installed Elasticsearch, Kibana and hosted in my localhost. Since the Google Colab cannot connect with localhost, I downloaded my saved DL models to local machine. To achieve my targets, I had to find a method to connect local python script which contained DL models with elastic search to receive and transfer data. My integration plan shows in Figure 4.1.19.

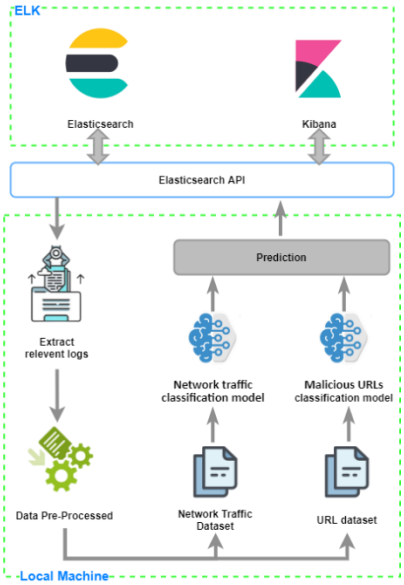


*Figure **3.1.4.7.1**: Real time data classification with ELK*

But here I have got another issue. When I tried to import my saved DL models to a local python script, I got error. Due to my local machine doesn't have a GPU and it

doesn’t fulfill the tensorflow and keras requirements I couldn’t import and run DL models in my local machine. If I configured the tensorflow to compatible with only CPU, it causes to reduce the efficiency of the DL models. Therefore, I decided to deploy and integrate my models in a cloud infrastructure as an API service and get prediction through the cloud.

3.1.4.8 Cloud API deployment

As mentioned before my next was to find a cloud infrastructure which provide services to deploy DL models. When finding a cloud infrastructure, I mainly focused about the cloud services contain facility for DL computation. I found that the Google Cloud Platform (GCP) [24] provides lot of services for AI technologies along with many other cloud services. The Vertex AI [25] is a new service which provides by GCP. It makes an environment to develop and maintain complete ML workflow as shown in figure 4.1.20. Also optimized to run inputs data through hosted AI models with little latency. Therefore, I decided to deploy my DL models on GCP Vertex AI.

Before get access for Vertex AI API, its need to create an account on GCP and create a new project. The project ID is the unique name across all GCP services. Next, enable the API services for the created project. For that I used the Cloud Shell. Cloud Shell is a command line interpreter which provided by GCP. The command used to enable API services.

To create endpoint on Vertex AI, DL models must have on the cloud. Therefore, I created a Cloud Storage Bucket and upload my DL models. After the models in bucket was imported to Vertex AI and deployed an endpoint for models by filling necessary fields. When deploying models, GCP provides functions to customize the cloud VM as we need. Also, it has function to configure GPU. The endpoint helps to get prediction from deployed models via the Vertex AI API. The model deployment workflow

To get prediction through the deployed API, the input must be transfer via a JSON request. And the Vertex AI defined a format for the JSON request too. Figure 4.1.23. Also, the prediction response came as a JSON format

Since my DL models were deployed on the cloud, they can be use from my local machine or any other machines via the endpoint call. To access the endpoint from a local machine, its need to be authenticated. Therefore, I created an authentication key by adding new role which has access to Vertex AI endpoint. Then the created key should be added as an environmental variable in the local machine. After these configurations I was able to get prediction for input data through a python script. Here it needed to be mentioned details of the endpoint such as project id, endpoint id. The endpoint call format through a python, shown in figure 4.1.25.

After all those processes, I integrate my local python script with ELK and classified real time data. That was the process for implementing deep learning models and deploy the trained model on cloud and integration with ELK for real time data analysis.

3.1.5 FUNCTIONING AND IMPLEMENTATION OF BLOCKCHAIN

In this section of the research, we will be looking at the functionalities of the blockchain.

- Orderly store the captured data.
- Ensuring the security, efficiency of the stored data in the blockchain.
- Integrity Check using Proof of Work
- Blockchain implementation in the cloud

The blockchain is used to secure store the outputs generated by the deep learning model and the NLP model. The data that is stored inside each block is composed with a timestamp, cryptographic hash of the previous block and a nonce value to maintain the security of each block.

3.1.5.1 ORDERLY STORING CAPTURED DATA

HOW BLOCKCHAIN STORAGE WORKS?

Blockchain relies on technology for distributed ledger (DLT). The DLT functions as a decentralized information database for transactions between different parties. The

procedure chronologically fills the DLT and is stored as several blocks in a ledger. Between blocks is built a linked chain, with each referencing the preceding block, establishing a blockchain.

Files are split down initially via a technique called sharding in the blockchain storage. In case of mistake during transmission, each shard is duplicated to prevent data loss. The data are additionally encrypted with a private key, which prevents the viewing of other network nodes. The duplicated shards are dispersed around the world through decentralized nodes. The interactions are logged in the blockchain directory, so that the system can confirm and sync the transactions across the blockchain nodes. Blockchain

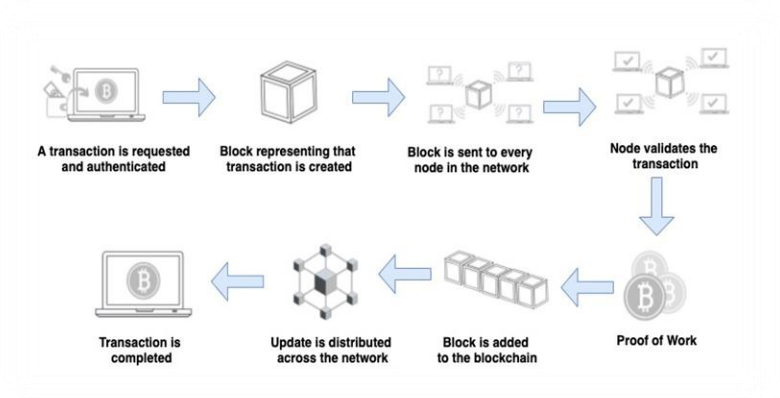


Figure 25Figure 3.1.5.1.1 Process of blockchain storage

Blockchain works via a multistep process, which in simple terms happens as follows:

1. An authorized participant inputs a transaction, which must be authenticated by the technology.
2. And when the transaction is accepted a new block is created. Which means for every transaction a new block is being created.
3. Then the block is sent to every node in the network.
4. Authorized nodes verify the transaction and add the block to the existing blockchain.

5. Proof of Work

6. Block is added to the Blockchain.

7. The update id distributed across the network which finalizes the transaction.

WHY USE BLOCKCHAIN TO STORE DATA?

- Probably the greatest advantage is security. A blockchain cannot be corrupted, because hundreds, even millions, of computers share and continuously reconcile information. There is also no single failure point at Blockchain.
- Transactions are more efficient than non-DLT-based transaction systems, however sluggish speed and inefficiency can occasionally be experienced by public blockchains.
- It's durable: No problem if one node falls down, because there is a duplicate of the ledger of all the other nodes.
- It creates confidence among network participants. It is highly difficult to reverse confirmed blocks which implies that data can be removed or modified.

- It can be economical since it frequently decreases transaction expenditure by removing intermediaries and third parties.

3.1.5.2 ENSURING THE SECURITY, EFFICIENCY OF THE STORED DATA IN THE BLOCKCHAIN

To provide security, ensure privacy and establish trust between entities, blockchain technology was introduced by Satoshi Nakamoto in 2008 [8]. It is an increasing list of records, stored in the form of blocks, which are joined together in a chronological order to form a chain using cryptographic hashes. Each block contains multiple components, such as a cryptographic hash of the previous block, timestamp at which block is generated, transaction data in the form of Merkle root tree and nonce, which is an arbitrary number used for mining process. The basic structure of the block is shown in Figure 1[11]. With the invention of blockchain, limitations of centralized architecture are eliminated. Blockchain ensures security, privacy preservation, decentralized architecture and data integrity. Due to the excellent features of blockchain, it is being used in various fields of life like Internet-of-Things (IoT) [9], smart grids [10], VNs, etc.

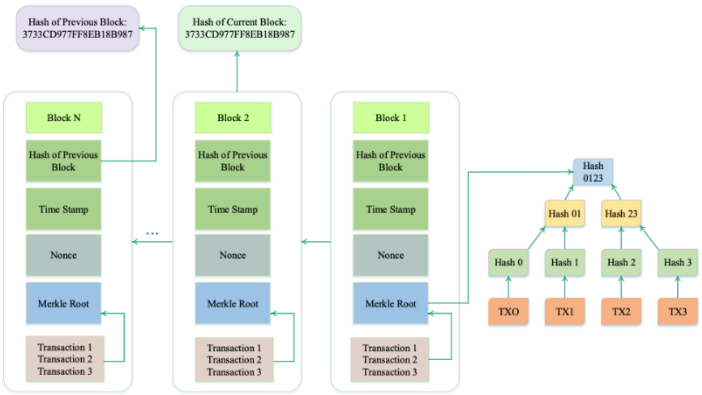


Figure 26Figure 3.1.5.2.1 Basic Structure of the Block

Figure 3.1.5.2.1 Basic Structure of the Block

If the hashing procedure is repeated with the same transactions, the same hashes are generated. This lets anybody who uses the blockchain to verify that data have not been manipulated, as any modification in any portion of the data will lead to a totally new hash, impacting all hash iterations to the root.

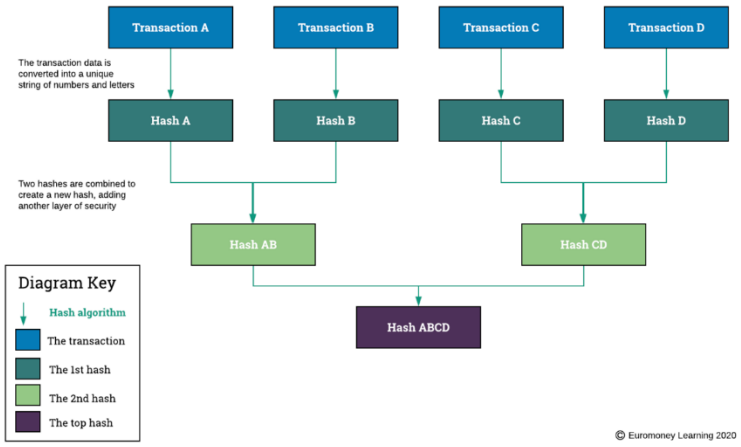


Figure 27Figure 3.1.5.2.2 Merkle Tree Diagram

Figure 3.1.5.2.2 Merkle Tree Diagram

Merkle Trees serve the purpose of significantly reducing the amount of data required to be stored and transmitted or broadcast over the network by summarizing sets of hashed transactions into a single root hash. As each transaction is hashed, then combined and hashed again, the final root hash will still be a standard size.

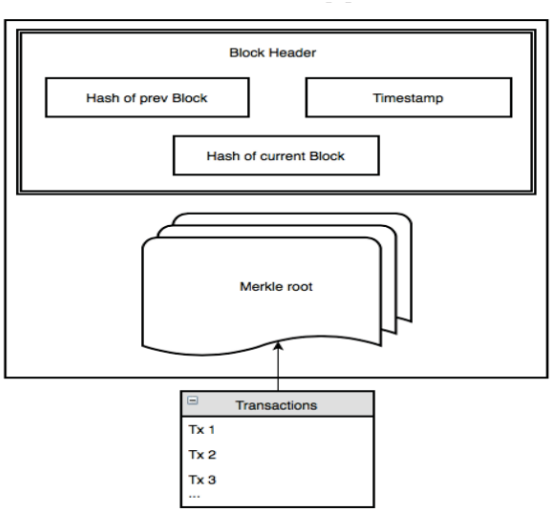


Figure 28Figure 3.1.5.2.3 Structure of a block with Merkle root

BLOCK CREATION

When creating a block of the block chain five variable were taken into consideration. The index number, the transaction data in the form of Merkle root tree, a time stamp (whenever a node is connected to a node, a UTC time stamp is taken from the node and the offset is saved in a UTC node), the nonce value (is a pseudo-random number used as a counter in the mining process), and the hash of the previous block.

```
class Block:

    def __init__(self, index, transactions, timestamp, previous_hash):
        self.index = index
        self.transactions = transactions
        self.timestamp = timestamp
        self.previous_hash = previous_hash
        self.nonce = 0
        self.hash = self.calculateHash()
```

Figure 29Figure 3.1.5.2.4 Implementation of Block

HASHING AND MERKLE TREE

Typically, a large list is to be taken and condensed in a short list. A long list is shown by the short list (hash) (input). Hashing is often used to accelerate search via programming and database. The hash function is termed the algorithm for generating hash.

A hashing algorithm is a math feature condensing data to a fixed size. Here, just the cryptographic hashing is of relevance for us. Hash is used as fingerprint for certain input data, known as message digest in cryptography. The hash will be different as the data changes. You can calculate if the data has changed in any manner by computing the Hash. Hash is used in. blockchain to represent data integrity of block.

```
def calculateHash(self):
    obj_dict = {
        'index': self.index,
        'transactions': self.transactions,
        'timestamp': self.timestamp,
        'previous_hash': self.previous_hash,
        'nonce': self.nonce
    }
```

Figure 30Figure 3.1.5.2.5 Implementation of Hash Calculation in Blockchain

The blockchain data are efficient and safe to encode in a hash tree or the Merkle tree. It provides fast blockchain verification of data and fast transport of huge quantities of data over a peer-to-peer blockchain network from one computer node to the other.

There is a hash for each transaction on the blockchain network. But these hashes are not saved in the block in sequence, but in the form of a tree-like structure that linked each hash to its parent according to a relationship of parent-child tree.

As many transactions are recorded on a given block, the entire block hashing, which lead to a Merkle root, are also hashed.

Merkle trees help in blockchain deleting duplicate or incorrect transaction data. For example, if someone tries to hack an entry on a blockchain and it seems to be more bitcoin than it really is, the false entry in a Merkle tree doesn't match the other dangers. This is because no transaction alone may be used when checking a blockchain transaction. All hash should be verified for every other hash to make sure nothing has been modified or changed.

Take a seven-transaction block, for example. Four transaction hashes are to be found at the lowest level (called the leaf-level). At level one above the leaf level there are two transaction hashes, each connecting to the two hashes below the leaf level. The final transaction hash (root) will appear at the top of level (level two) and will be linked to the next two hashes (at level one).

You obtain an upside-down binary tree with just two nodes below each node of the tree (hence the name "binary tree"). It has one hash root at the top, connecting two hashes at level one and connecting to the two hashes at level three again (leaf level), and the structure continues according to the number of hashes.

The hashing starts at the lowest node (leaf level), and the four hashes of nodes related to this level one is included in the hash. similarly, hashing at level 1, which results in hashes that reach higher levels until the one top root hash is reached.

The root of Merkle which speeds up the verifications. Given that the whole tree is fully informed, just the transaction hash, its sibling node (if any), must be checked and then upward up to the top.

Merkle and the Merkle root method essentially dramatically lower hash levels, allowing for quicker verification and transactions.

The root hash is known as the root of Merkle, and because of the tree like connection of hashes, it holds all the information on each hash on the block. It gives a unique hash value that allows all the present on that block to be validated.

```
class Merkleroot(object):
    def __init__(self):
        pass

    def doubleSha256(input):
        return hashlib.sha256(hashlib.sha256(input).hexdigest()).hexdigest()

    def findMerkleRoot(self, leafHash):
        hash = []
        hash2 = []
        if len(leafHash) % 2 != 0:
            leafHash.extend(leafHash[-1:])

        for leaf in sorted(leafHash):
            hash.append(leaf)
            if len(hash) % 2 == 0:

                hash2.append(doubleSha256(hash[0]+hash[1]))
                hash == []
        if len(hash2) == 1:
            return hash2
        else:
            return self.findMerkleRoot(hash2)
```

Figure 31Figure 3.1.5.2.6 Implementation of Merkle Root in Blockchain

In figure above shows how the Merkle root is implemented in the blockchain. Here two hashes are being implemented as this is the Merkle root. As you can see the remainder should be zero when you divide the length of the leaf hash by 2. And if it's not even the last element should be repeated. And for each leaf only add secondary hash if there are two first hashes if that condition is satisfied run through the hash function for both hashes (Double hashes). Then reset the first hash to empty, if the secondary hash is only one, we are the root if not recurse with hash2.

3.1.5.3 INTEGRITY CHECK USING PROOF OF WORK

Proof of work or PoW in a Blockchain network is the original consensus algorithm. This method is used in Blockchain to validate transactions and create new blocks on the chain. With PoW, miners compete to finish network transactions. Digital tokens are sent in a network user. All transactions are grouped into blocks by a decentralized ledger. However, the transactions should be confirmed, and blocks organized. The main working principles are a complicated mathematical puzzle and a possibility to easily prove the solution.

The amount of users, current capacity and network demand depend on how hard a puzzle is. The hash of each block contains the hash of the preceding block, which enhances safety and avoids violations of blocks.

The new block is created if a miner solves this puzzle. The transactions are placed and verified in this block.

Why use a Proof of Work algorithm in the blockchain?

The primary advantages are the resistance against DoS attacks and the little effect on mining options. DoS assaults in defense. PoW sets some limitations on network operations. You must make a lot of effort. A powerful assault needs a great deal of processing power and time to calculate. The assault is therefore viable, albeit rather ineffective, as the costs are too high.

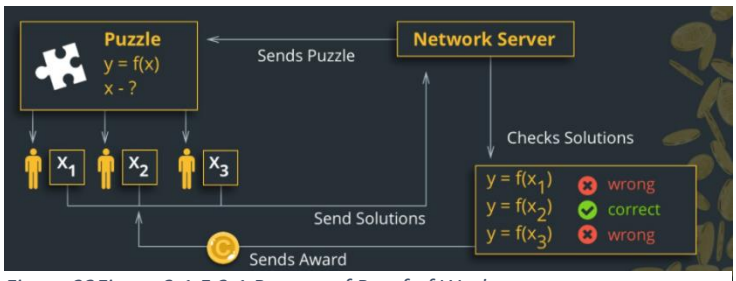


Figure 32Figure 3.1.5.3.1 Process of Proof of Work

```
def proofOfWork(self, block, difficulty):
    block.nonce = 0
    calculatedHash = block.calculateHash()
    while not calculatedHash.startswith('0' * difficulty):
        block.nonce += 1
        calculatedHash = block.calculateHash()
    return calculatedHash
```

Figure 33Figure 3.1.5.3.2 Implementation of PoW in Blockchain

According to the above figure, it shows how Proof of Work is implemented in the blockchain. As you can see the initial value of nonce in the block is zero (0). And then it calculates the hash value for with the nonce value at zero. After the hash value is calculated we are using a while loop to check whether the calculated hash value is satisfying with the Pow constraint, we declared which is the block hash shouldn't be starting with n number of leading zeros. And if the constraint is not satisfied the nonce value will be increased by one (+1) and then calculate the hash value. The hash value is calculated while increasing the nonce value until the constraint is satisfied.

```
# Integrity check
def isChainValid(self):
    for i in range(1, self.chain.length):
        currentBlock = self.chain[i]
        previousBlock = self.chain[i - 1]

        if currentBlock.hash != currentBlock.calculateHash():
            return False

        if currentBlock.previousHash != previousBlock.calculateHash():
            return False

    return True
```

Figure 34Figure 3.1.5.3.3 Integrity Check in Blockchain

3.1.5.4 BLOCKCHAIN IMPLEMENTATION IN THE CLOUD

Data is split into many encrypted areas with a hyperlink feature in closed based blockchain. These secure segments are dispersed throughout the network and each section is decentralized. Strong safety provisions like as transaction ledgers, pubic/private key encryption and hashed blocks exist. It provides dependable and

strong hacker security. Due to strong 256-bit encryption, even skilled hackers can't decipher the data.

The dispersed cloud storage enables customers with a protected P2P network to store their mission-critical data decentralised without jeopardizing their security. The use of Blockchain technology to store your data has additional tremendous advantages. It is 10 times quicker and the overall expenditures can be reduced to half. Strategically, distributed club storage combines transparency and security using the hash function, public/private key encryption and safe transaction headers.The decentralized cloud-based storage also safeguards customers from various security concerns.

Blockchain offers full transparency, just to prevent any nefarious behavior on the network from being shaded anymore. In addition, the sequential storage assures that every transaction can always be validated. The Blockchain-based cloud storage also allows immuable transaction records to check ownership and identity through the creation of a robust, structured and linked mesh of block.

```
[
  {
    "": 0,
    "ID": 1,
    "host": "www.ird.gov.lk",
    "url_label": "good",
    "traffic_label": "good",
    "user_agent": "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:92.0) Gecko/20100101 Firefox/92.0",
    "id_resp_p": 80,
    "method": "GET",
    "uri": "/_layouts/15/ramis/img/language-logo-text.png",
    "version": 1.1,
    "@timestamp": "2021-09-22T09:20:34.438Z",
    "status_msg": "Not Modified",
    "response_body_len": 0,
    "url": "www ird gov lk  layouts 15 ramis img language logo text png",
    "feature": "GET 80 1 1 www ird gov lk  layouts 15 ramis img language logo text png Mozilla 5 0
        X11  Ubuntu  Linux x86 64  rv 92 0  Gecko 20100101 Firefox 92 0 Not Modified 0"
  },
  {
    "": 1,
    "ID": 2,
    "host": "www.ird.gov.lk",
    "url_label": "good",
```


Figure 35Figure 3.1.5.4.1 Post Request with the Output send to the Cloud Server

The *Figure 4.1.4.4.2* shows the request sent to the Heroku server through POSTMAN to save the results generated by the Neuro Realistic Detection Engine and AI engine in the blockchain in the Heroku cloud server.

```
},
{
    "": 9,
    "ID": 10,
    "host": "www.statistics.gov.lk",
    "url_label": "good",
    "traffic_label": "good",
    "user_agent": "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:92.0) Gecko/20100101 Firefox/92.0",
    "id_resp_p": 80,
    "method": "GET",
    "uri": "/img/slider/CCPI.jpg",
    "version": 1.1,
    "@timestamp": "2021-09-22T09:20:40.372Z",
    "status_msg": "Not Modified",
    "response_body_len": 0,
    "url": "www statistics gov lk img slider CCPI jpg",
    "feature": "GET 80 1 1 www statistics gov lk img slider CCPI jpg Mozilla 5 0 X11 Ubuntu Linux
        x86 64 rv 92 0 Gecko 20100101 Firefox 92 0 Not Modified 0"
}
```


Figure 36Figure 3.1.5.4.2 Post Request with the Output send to the Cloud Server

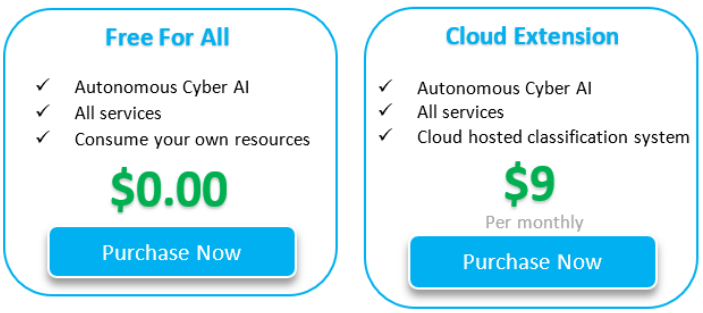

```
        "ID": 1,
181     "host": "www.ird.gov.lk", "url_label": "good", "traffic_label": "good", "user_agent": "Mozilla/5.0 (X11;
            Ubuntu; Linux
182     x86_64; rv:92.0) Gecko/20100101 Firefox/92.0", "id_resp_p": 80, "method": "GET", "uri":
183     "/_layouts/15/ramis/img/language-logo-text.png", "version": 1.1, "@timestamp": "2021-09-22T09:20:34.
            438Z", "status_msg":
184     "Not Modified", "response_body_len": 0, "url": "www ird gov lk layouts 15 ramis img language logo text
            png", "feature":
185     "GET 80 1 1 www ird gov lk layouts 15 ramis img language logo text png Mozilla 5 0 X11 Ubuntu Linux x86
            64 rv 92 0 Gecko
186     20100101 Firefox 92 0 Not Modified 0"}, "timestamp": 1634068802.8288474, "previous_hash":
187     "2bf7cc48b1945ec4d2ac658222be58de0da480fd08fafb263839d1046085ec1d", "nonce": 0, "hash":
188     "2bf7cc48b1945ec4d2ac658222be58de0da480fd08fafb263839d1046085ec1d"}, {"index": 2, "transactions": {"": 1,
            "ID": 2,
189     "host": "www.ird.gov.lk", "url_label": "good", "traffic_label": "good", "user_agent": "Mozilla/5.0 (X11;
            Ubuntu; Linux
190     x86_64; rv:92.0) Gecko/20100101 Firefox/92.0", "id_resp_p": 80, "method": "GET", "uri": "/_layouts/15/
            ramis/img/ta.png",
191     "version": 1.1, "@timestamp": "2021-09-22T09:20:34.439Z", "status_msg": "Not Modified",
            "response_body_len": 0, "url":
192     "www ird gov lk layouts 15 ramis img ta png", "feature": "GET 80 1 1 www ird gov lk layouts 15 ramis img
            ta png Mozilla
193     5 0 X11 Ubuntu Linux x86 64 rv 92 0 Gecko 20100101 Firefox 92 0 Not Modified 0"}, "timestamp": 1634068802.
            8330517.
```


Figure 37Figure 3.1.5.4.3 Post Response

3.3 Commercialization Aspects of the Product

Most of the SIEM products available in the market are a dream to buy for most of the information security providing service companies due to its high cost. There small and medium scale business cannot afford to buy such high price products. Due to the above reason we planned to release an open source product which is an SIEM to facilitate even to the small and medium scale businesses. These vendors can also use this SIEM to detect threats, work on threat intelligence and integrate this product to their security framework. This project can also be found in open source platforms so that vendors can do their desired changes according to their security requirement.

This product is focusing on human free involvement since its artificial intelligence engine is trained using deep learning models where it has been trained using custom generated datasets. This product checks for all malicious activities in the network including nmap scans, dos attacks, malicious IP recognition, malicious URL recognition, and detects malware in the network. This product fits to critical assets of any organization since those are the assets which needs to be kept confidential and which contains sensitive information. By considering all these features small and medium scale businesses does not have to rely on a SOC, this product can be implemented easily and protect critical assets of these businesses.



4. Testing and Implementation.

This research focuses on capturing real time data for anomaly detection. In order to achieve the task, we have used mechanism like filebeat, packetbeat, zeek, cicflowmeter etc. This process starts with data capturing using beats and then filtering it. In order to make this product very efficient and effective we have to capture the needed fields required for the Artificial intelligence engine and the Neuro realistic Engine. The two AI based engines in the autonomous cyber AI uses different ML models which uses different datasets. It is the responsibility of the threat detection engine to capture the relevant data for the AI engine and the Neuro realistic engine to conduct malicious analysis on the data passed to them.

According to the research the Artificial intelligence engine will be using ML models to train its data, in order to train the data, it is using a dataset known as CICIDS2017 which contains 73 features, in order to extract these features. In order to capture these 73 features, we had to put a significant effort in finding a mechanism which supports CICIDS2017 dataset. In order to extract these 73 features, we used a tool called CICFlowmeter which is used to convert the captured data into these 73 features and produce a csv output where it is passed to the next phase using the filebeat.

Next moving on to the neuro realistic engine it uses another dataset where its features where able to capture using zeek and put into a json format.

This part of the research is done to capture the logs for the AI engine. Following is the obtained dataset.

src_ip	dst_ip	src_port	dst_port	protocol	timestamp	flow_duration	flow_byts_s	flow_pkts_s	fwd_pkts_s	bwd_pkts_s	tot_fwd_pkts	tot_bwd_pkts	totl
192.168.85.1	192.168.85	47275	53	17	2021-09-19 02:	56865	3095.0496791	35.171019080278	17.5855095401389	17.5855095401389	1	1	
192.168.85.1	192.168.85	42238	53	17	2021-09-19 02:	57417	3274.2915861	34.832889213996	17.4164446069979	17.4164446069979	1	1	
192.168.85.1	192.168.85	45665	53	17	2021-09-19 02:	56806	3626.3774953	35.207548498398	17.603774249199	17.603774249199	1	1	
192.168.85.1	192.168.85	39406	53	17	2021-09-19 02:	57416	3796.851052	34.833495889648	17.4167479448237	17.4167479448237	1	1	
192.168.85.1	192.168.85	50064	53	17	2021-09-19 02:	6364	69138.906348	314.26775612822	157.133878064111	157.133878064111	1	1	
192.168.85.1	192.168.85	41682	53	17	2021-09-19 02:	7049	41991.771883	283.72818839552	141.864094197759	141.864094197759	1	1	
192.168.85.1	192.168.85	47272	53	17	2021-09-19 02:	57727	6894.5207615	34.645832972439	17.3229164862196	17.3229164862196	1	1	
192.168.85.1	192.168.85	50197	53	17	2021-09-19 02:	58335	4354.1613097	34.284734721865	17.1423673609325	17.1423673609325	1	1	
192.168.85.1	142.250.76	35724	80	6	2021-09-19 02:	12229312	37.28746147	0.6541659906951	0.327082995347571	0.327082995347571	4	4	
192.168.85.1	192.168.85	40386	53	17	2021-09-19 02:	61697	3079.5662674	32.416487025301	16.2082435126505	16.2082435126505	1	1	
192.168.85.1	192.168.85	47519	53	17	2021-09-19 02:	61085	3306.8674797	32.741262175657	16.3706310878284	16.3706310878284	1	1	
192.168.85.1	192.168.85	52821	53	17	2021-09-19 02:	51093	3855.71409	39.14430548216	19.57215274108	19.57215274108	1	1	
192.168.85.1	192.168.85	54053	53	17	2021-09-19 02:	51495	4058.6464705	38.83872220604	19.4193611030197	19.4193611030197	1	1	
192.168.85.1	192.168.85	50302	53	17	2021-09-19 02:	52466	3735.7526779	38.119925284947	19.0599626424732	19.0599626424732	1	1	
192.168.85.1	192.168.85	45412	53	17	2021-09-19 02:	52322	3975.383204	38.224838500057	19.1124192500287	19.1124192500287	1	1	
192.168.85.1	192.168.85	54507	53	17	2021-09-19 02:	51554	3821.2359856	38.794273965163	19.3971369825814	19.3971369825814	1	1	
192.168.85.1	192.168.85	54424	53	17	2021-09-19 02:	52482	3982.317747	38.108303799398	19.0541518996989	19.0541518996989	1	1	
192.168.85.1	192.168.85	38717	53	17	2021-09-19 02:	50122	3272.0162803	39.902637564343	19.9513187821715	19.9513187821715	1	1	
192.168.85.1	192.168.85	59637	53	17	2021-09-19 02:	50393	3492.5485683	39.688051911972	19.844025955986	19.844025955986	1	1	
192.168.85.1	192.168.85	59511	53	17	2021-09-19 02:	32263	5021.2317515	61.990515451136	30.995257725568	30.995257725568	1	1	
192.168.85.1	192.168.85	53928	53	17	2021-09-19 02:	35601	4887.50316	56.178197241651	28.0890986208253	28.0890986208253	1	1	
192.168.85.1	192.168.85	43275	53	17	2021-09-19 02:	29941	5343.842891	66.798036137738	33.3990180688688	33.3990180688688	1	1	
192.168.85.1	192.168.85	43676	53	17	2021-09-19 02:	44771	3841.772576	44.67177413951	22.335887069755	22.335887069755	1	1	
192.168.85.1	69.171.250	54961	443	17	2021-09-19 02:	398593	22393.770086	50.176495824061	20.0705983296245	30.1058974944367	8	12	
192.168.85.1	69.171.250	46212	443	6	2021-09-19 02:	73779	30794.67057	108.43193862752	54.2159693137614	54.2159693137614	4	4	
192.168.85.1	192.168.85	44931	53	17	2021-09-19 02:	69559	2975.8909703	28.752569760922	14.3762848804612	14.3762848804612	1	1	
192.168.85.1	192.168.85	42874	53	17	2021-09-19 02:	69982	3129.3761253	28.578777399903	14.2893886999514	14.2893886999514	1	1	
192.168.85.1	192.168.85	42561	53	17	2021-09-19 02:	158421	1256.1465967	12.624588911824	6.31229445591178	6.31229445591178	1	1	
192.168.85.1	192.168.85	59752	53	17	2021-09-19 02:	158271	1522.704728	12.636553759059	6.31827687952941	6.31827687952941	1	1	

Figure 38Figure 4.1 output of cicflowmeter

The above output will be passed to logstash using filebeat and csv filter will be used to break the values into each field in the elasricsearch.

When moving on to the neuro realistic engine for mainly URL detections the following datasets were used.


```
{"ts":1633356013.55152,"uid":"Cswd8i3akRperlxqZ8","id.orig_h":"192.168.85.137","id.orig_p":34224,"id.resp_h":"117.18.237.29","id.resp_p":80,"trans_depth":1,"method":"POST","host":"ocsp.digicert.com","uri":"/","version":"1.1","user_agent":"Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:92.0) Gecko/20100101 Firefox/92.0","request_body_len":83,"response_body_len":278,"status_code":200,"status_msg":"OK","tags":[],"orig_fuids":["F7FpdA46dkhBZ1u7M9"],"orig_mime_types":["application/ocsp request"],"resp_fuids":["FKzEhT3vZ0do7WnRhl"],"resp_mime_types":["application/ocsp-response"]}
{"ts":1633356024.890484,"uid":"CdHxyz1hsx0MBHT6G3","id.orig_h":"192.168.85.137","id.orig_p":58402,"id.resp_h":"125.214.166.51","id.resp_p":80,"trans_depth":3,"method":"POST","host":"r3.o.lencr.org","uri":"/","version":"1.1","user_agent":"Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:92.0) Gecko/20100101 Firefox/92.0","request_body_len":85,"response_body_len":503,"status_code":200,"status_msg":"OK","tags":[],"orig_fuids":["F5hhDS14qcXy4QYiEl"],"orig_mime_types":["application/ocsp request"],"resp_fuids":["Fe3TB7dgf22Bgh7x9"],"resp_mime_types":["application/ocsp-response"]}
{"ts":1633356083.943526,"uid":"CIavpw3GWCYizyuHZa","id.orig_h":"192.168.85.137","id.orig_p":59186,"id.resp_h":"50.63.7.226","id.resp_p":80,"trans_depth":1,"method":"GET","host":"maliciouswebsitetest.com","uri":"/","referrer":"https://www.google.com/","version":"1.1","user_agent":"Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv 92.0) Gecko/20100101 Firefox/92.0","request_body_len":0,"response_body_len":2858,"status_code":200,"status_msg":"OK","tags":[],"resp_fuids":["FIovLn2uip3N8MxaDd"],"resp_mime_types":["text/html"]}
{"ts":1633356084.618826,"uid":"C0QfkH1lYqJeQLtqEh","id.orig_h":"192.168.85.137","id.orig_p":59188,"id.resp_h":"50.63.7.226","id.resp_p":80,"trans_depth":1,"method":"GET","host":"maliciouswebsitetest.com","uri":"/favicon.ico","referrer":"http://maliciouswebsitetest.com/","version":"1.1","user_agent":"Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:92.0) Gecko/20100101 Firefox/92.0","request_body_len":0,"response_body_len":315,"status_code":404,"status_msg":"Not Found","tags":[],"resp_fuids":["FIUwmf4syW8CXvM0dk"],"resp_mime_types":["text/html"]}
{"ts":1633356089.543247,"uid":"C4s93hcVNfTpXmWYk","id.orig_h":"192.168.85.137","id.orig_p":42342,"id.resp_h":"5.61.58.161","id.resp_p":80,"trans_depth":1,"method":"GET","host":"gepardmain.com","uri":"/admin/","version":"1.1","user_agent":"Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:92.0) Gecko/20100101 Firefox/92.0","request_body_len":0,"response_body_len":381,"status_code":401,"status_msg":"Unauthorized","tags":[],"resp_fuids":["F5xoov4v0sOzXKNvI7"],"resp_mime_types":["text/html"]}
{"ts":1633356096.102333,"uid":"CX42wr2xEH6gjRcDRi","id.orig_h":"192.168.85.137","id.orig_p":33076,"id.resp_h":"185.215.113.20","id.resp_p":80,"trans_depth":1,"method":"GET","host":"185.215.113.20","uri":"/gb9fskvS/login.php","version":"1.1","user_agent":"Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:92.0) Gecko/20100101 Firefox/92.0","request_body_len":0,"response_body_len":3551,"status_code":200,"status_msg":"OK","tags":[],"resp_fuids":["FZnJp93Mwg3qTZqDHj"],"resp_mime_types":["text/html"]}
{"ts":1633356204.517245,"uid":"C2mlEq2IvHAPIRrPRk","id.orig_h":"192.168.85.132","id.orig_p":60804,"id.resp_h":"192.168.85.137","id.resp_p":80,"trans_depth":1,"method":"GET","host":"192.168.85.137","uri":"/DVWA/vulnerabilities/sqli/","referrer":"http://192.168.85.137/DVWA/vulnerabilities/xss_s/","version":"1.1","user_agent":"Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0","request_body_len":0,"response_body_len":4242,"status_code":200,"status_msg":"OK","tags":[],"resp_fuids":["FuKaYfeYJGDpPwVR2"],"resp_mime_types":["text/html"]}
```


Figure 39Figure 4.2 output for malicious URL detection.

In order to prove our result, some testing attacks were done to create datasets and train with the AI model, in this case the threat detection engine was responsible to create the needed datasets. Therefore, attacks like nmap, dos-hulk, sql injection, ftp predator and many more. The following diagrams show the attacks that has been done for the testing purposes.

```
root@ubuntu:/home/ubuntu/hulk/slowloris# python3 slowloris.py -p 80 -s 1000 -v 192.168.255.132
[06-10-2021 08:36:00] Attacking 192.168.255.132 with 1000 sockets.
[06-10-2021 08:36:00] Creating sockets...
[06-10-2021 08:36:00] Creating socket nr 0
[06-10-2021 08:36:04] timed out
[06-10-2021 08:36:04] Sending keep-alive headers... Socket count: 0
[06-10-2021 08:36:04] Recreating socket...
[06-10-2021 08:36:08] timed out
[06-10-2021 08:36:08] Sleeping for 15 seconds
[06-10-2021 08:36:23] Sending keep-alive headers... Socket count: 0
[06-10-2021 08:36:23] Recreating socket...
```

Figure 40Figure 4.3 slowloris attack

The slowloris attack is a type of dos attack which blocks the usage of a site or server for a period. The result of this is shown in the result section of this report.

```
root@ubuntu:/home/ubuntu/hulk# python hulk.py http://192.168.85.137/DVWA/login.php
-- HULK Attack Started --
1646 Requests Sent
1747 Requests Sent
1850 Requests Sent
1956 Requests Sent
2060 Requests Sent
2165 Requests Sent
2266 Requests Sent
2368 Requests Sent
2469 Requests Sent
2571 Requests Sent
2672 Requests Sent
2773 Requests Sent
2874 Requests Sent
2976 Requests Sent
3078 Requests Sent
3179 Requests Sent
3280 Requests Sent
3381 Requests Sent
3484 Requests Sent
3590 Requests Sent
3694 Requests Sent
3795 Requests Sent
3898 Requests Sent
4001 Requests Sent
4103 Requests Sent
4207 Requests Sent
4309 Requests Sent
4410 Requests Sent
```

Figure 41Figure 4.4 Dos hulk attack

```
root@ubuntu:/home/ubuntu/hulk# slowhttptest -c 1000 -X -i 10 -r 200 -t GET -u http://192.168.85.137/DVWA/login.php -x 24 -p 80
Wed Oct  6 08:33:04 2021:


Wed Oct  6 08:33:04 2021:
       slowhttptest version 1.6
 - https://code.google.com/p/slowhttptest/ -
test type:                        SLOW READ
number of connections:            1000
URL:                              http://192.168.85.137/DVWA/login.php
verb:                             GET
receive window range:             1 - 512
pipeline factor:                  1
read rate from receive buffer:    5 bytes / 1 sec
connections per seconds:          200
probe connection timeout:         80 seconds
test duration:                    240 seconds
using proxy:                      no proxy

Wed Oct  6 08:33:04 2021:
slow HTTP test status on 0th second:

initializing:         0
pending:              1
connected:            0
error:                0
closed:               0
service available:    YES
Wed Oct  6 08:33:09 2021:
```

Figure 42Figure 4.5 slowhttptest attack

These are some of the attacks carried out in the testing process of the research. By doing these testing it was able to obtain the needed results to our research.

4.1 Implementation process of the AI Engine

The threat intelligence engine has the capability of, capture real time traffic and data from multiple sources. This data capture engine is used to make accurate decisions on anomaly-based intrusions. And it collects data and network traffic real time from the host and the network. This system is focused to collect every data in the host system and network traffic making sure not to miss any important data for anomaly detection.

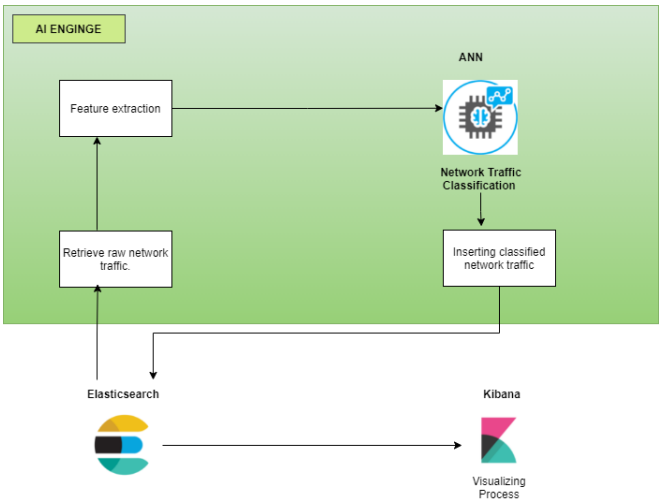


Figure 43Figure 4.1.1: High level overview of the AI Engine

Once the captured data has been filtered using Logstash, it is passed for further analysis to the AI engine. As it's shown in the above image AI engine's proposed to classify the raw network traffic related information passed to the Elasticsearch by the data capturing function.

Then with the help of joblib library scaler model was loaded which was previously saved while training the neural network model and it is used to standardize the features of the network traffic passed to the AI engine in raw format. After that the pre trained ANN was loaded using the keras load model instance. By default the loaded the ANN model would be compiled in this phase since the original ANN model was built and saved along with the optimizer. Then an instance of elastic client is created using Elasticsearch library to initiate a connection with the local Elasticsearch which was running on localhost: port 9200.

```
#load scaler model
sc= load('C:\scaler.bin')

#load ANN
ANN = tf.keras.models.load_model('C:\ANN.h5')

elastic_client = Elasticsearch()
```

Figure 44Figure 4.1.2: Loading scaler model and ANN

```
def main():

    count = 1
    while True:
        print('Records :', count)

        #retrive data and convert to dataframe
        Docs = get_elk_docs()

        # create a Pandas DataFrame array from the fields dict
        elastic_df = pd.DataFrame(Docs)

        #predict usning ANN
        Elastic_df = ann_predicts(elastic_df)

        # Exporting Pandas Data to Elasticsearch
        Elastic_df.insert(0, 'ID', range(count, count + len(Elastic_df)))
        df_iter = Elastic_df.iterrows()
        index, document = next(df_iter)
        helpers.bulk(es_client, ann_doc_generator(Elastic_df))

        print('Records :', count , 'exported to ELK \n')
        print('************************************')
        count = count + len(Elastic_df)
        # get new records in every 5 seconds
        time.sleep(5)

if __name__ == '__main__':
    main()
```

Figure 45Figure 4.1.3: Main function of AI engine

After those preliminary steps under main function of the AI engine it initially calls the get_elk_docs function which was utilized to retrieve raw traffic records from the index which comprise of raw traffic.

There it retrieves the documents from the specified index. (In this case the index is “test”) using the API calls to the Elasticsearch and stores the returned response object which is a document in JSON format in a variable. There the first “hits” represents the number of documents obtained via the API call and the second “hit” contains the actual raw data corresponds to each document and enumerate function is used to iterate through each document obtained through API call. Following image visualizes the similar output returned by the response object and represents the key difference between two “hits” utilized in the response. Then filters out the specified actual data from the “_source” and returns a dictionary after the function execution process.

In the next phase the returned dictionary converted in pandas' data frame and passed in to the ann_predict function where the ANN used to classify each record based on their features.

```
def ann_predicts(elastic_df):
    F_df = elastic_df
    f_df = F_df.drop(['src_ip', 'dst_ip'], axis = 1)
    features = sc.transform(f_df)
    preds = ANN.predict(features)
    predictions = np.argmax(preds, axis=1)
    labels = predictions.astype(str)

    for i in range(labels.shape[0]):
        if(labels[i] == '0'):
            labels[i] = 'benign'
        elif(labels[i] == '1'):
            labels[i] = 'bot'
```

Figure 46Figure 4.1.4: Classification process of the AI engine

In here the original data frame which is sent to the function contains the source IP address and destination IP address because when the records are visualized through interface it should be able to understand the originating source and the destination source by any user. But when it comes to ANN those sockets related information was removed in the preprocessing steps before training to deliver unbiased robust prediction. Therefore, here also initially those IP related columns are dropped and data frame which contains pure feature set is passed into the model for classification purposes. Then it re appends the corresponding attack label to each record in the original data frame and return it after the execution process. In the final stage the data frame is converted back to documents with the help of following function shown in the image below and passed back into the specified index (in this case "elk_test") in the Elasticsearch.

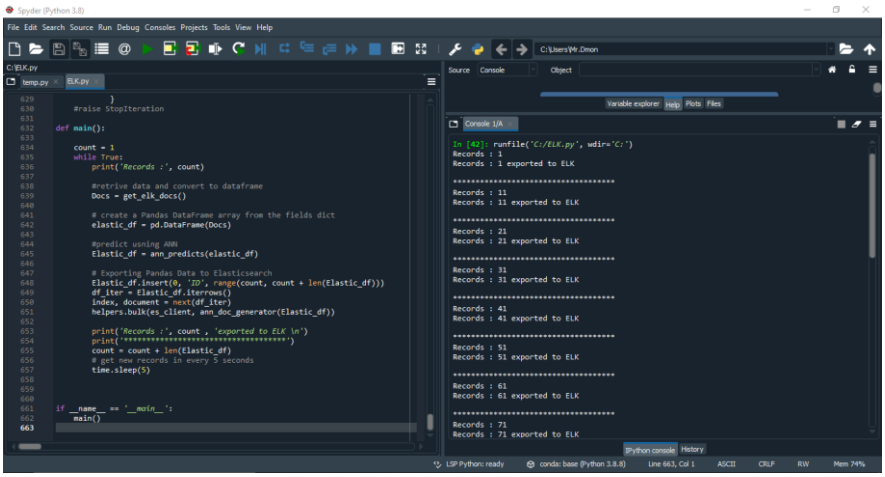


Figure 47Figure 4.1.5: Execution process of the AI Engine

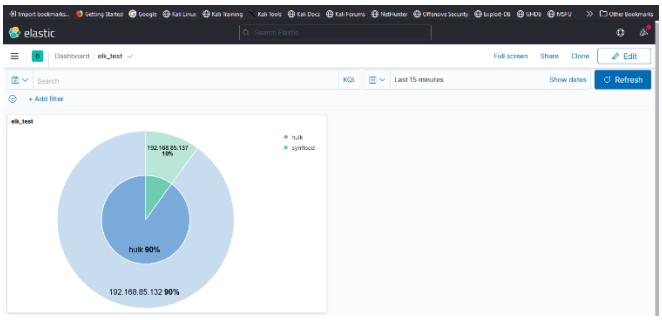


Figure 48Figure 4.1.6: Visualize the index created by AI Engine using Kibana

When it comes to the testing phase Kali Linux operating system and Ubuntu operating system installed in VM ware workstation and vmnet was created to isolate both machines. Then Kali Linux machine used to carry out different attack against Ubuntu machine while ensuring the integrity of the network traffic.

4.2 Testing of the Neuro Realistic Detection

The model testing and evaluation acts a major part of a research. From model evaluation we can decide major components of deep learning models, model accuracy and model loss that was the initial configuration part for the whole research component scenario. For initial model testing and evaluation, I chose Google Colab as my testing environment with the hardware specification which shown in table 4.2.1

Table 1Table 4.2.1: Google Colab hardware specifications

Parameter	Google Colab
CPU Model	Intel(R) Xeon(R)
CPU Frequency	2.30GHz
GPU	Nvidia Tesla K80
GPU Memory	12 GB
GPU Memory Clock	0.82GHz / 1.59GHz
RAM	13 GB
Disk Space	74 GB

Test the Network Traffic Classification Model with Both Dataset

Table 2Table 4.2.2: Test case 01

Test case ID	01
Test case scenario	Train and test Network Traffic Classification model using HTTP CSIC 2010 dataset
Test steps	i). Pre-process the data ii). Combine the dataset iii) Train the word2vec model iv). Split the dataset for training and testing v). Train the model and retrieve training accuracy vi). Test the model and retrieve the accuracy
Expected Outcome	More than 80% accuracy

In this test scenario, I firstly imported the HTTP CSIC 2010 dataset and our dataset. After done the pre-processing, split both datasets as train and test separately. After that

merged CSIC' train dataset with ours train dataset, like vice test datasets. Then trained the word2vec models and created the embedding matrix for neural network. Next, started to train the model using the same parameters which used for previous training phases. After training and visualizing the outputs, inputted the test dataset for prediction. Then tested out and printed the results.

Since this trained model give good accuracy comparing with other two tested model, I decided to choose the merged dataset as my final dataset for training this classification model. Therefore, I trained the network traffic classification model with 250 epochs, 128 batch size using merged dataset. It took about 3.30 hours to complete the training

Test Malicious URLs Classification Model with Both Dataset

Table 3Table 4.2.3: Test case 02

Test case ID	02
Test case scenario	Train and test Malicious Traffic Classification model using Malicious URLs dataset and our dataset
Test steps	i). Pre-process the data ii). Combine the dataset iii) Train the word2vec model iv). Split the dataset for training and testing v). Train the model and retrieve training accuracy vi). Test the model and retrieve the accuracy
Expected Outcome	More than 80% accuracy

After training and testing network traffic classification model, I started to train and test malicious URLs classification model. In this test case, I also made a new dataset by merging Malicious URL dataset with our dataset. Then cleaned the texts using same algorithm used before and trained the word2vec models for create an embedding

metrics. After that, split the dataset as 80% for training and other 20% for testing. Next the tokenized train data went through the neural network under 250 epochs, and it took about 3 hours. Then visualize the training results, also the testing results which generated by predicting with test data. And the results were fulfilled my expected results.

Test vertex AI API Response

Table 4Table 4.2.4: Test case 03

Test case ID	03
Test case scenario	Test the Vertex AI API prediction responses during Peak and Off-Peak hours.
Test steps	i). Pre-processed the request data ii) Initialize time capturing iii) Make prediction request to model iv) Capture the model's response v) Analyze response and the latency
Expected Outcome	Accurate predictions with low latency

End of model testing another most important component of my research work was deploy the trained DL models on a cloud service and get response through the cloud. Response time is most important factor in intrusion detection because user notification depends on this response time if any latency occurs while sending response then user get notified with delay then it can be big issue user is not able to get any action about detected malicious activities.

The network latency is not stabled in whole day since in peak hours (08.00 to 00.00) network traffic load is high and in off-peak hours (00.00 to 08.00) do not have that much of traffic. Therefore, I decided to measure response time by get response in random times in peak hours and off-peak hours.

First, I randomly selected 10 samples from our dataset since we used same method to capture real time data. Then pre-processed them and tokenized them to compatible with trained models' inputs. Here I made a python function to connect with the Vertex AI API and make JSON requests and retrieved the model response. Furthermore, the function converts the prediction outputs into labels and display them. Then using this function, I made requests and get responses during peak and off-peak hours while calculating the time. From this test case expected less than 10 response time for peak hours and less than 7 time for off-peak hours from 1 response record.

Test Real Time Data Classification with ELK Stack

Table 5Table 4.2.7: Test case 04

Test case ID	04
Test case scenario	Test classification and visualizing real time data with ELK stack
Test steps	i). Retrieve the traffic data from Elastic search ii) Pre-process the data iii) Make prediction through Vertex AI API iv) Export the results to Elastic search v) Visualize the using Kibana
Expected Outcome	Real time data classification with visualizing

My final testing and implementation stage was integrating the Neuro Realistic Detection System with ELK stack for analyze real time data. And it was the main objective of my research works. Therefore, I coded a python script which retrieves the network logs through Elastic search API and convert them into data frames as described in the methodology. After preprocessing those data, inputted into the python function which mentioned in test case 05 for get prediction from cloud hosted DL

models. Before send the labeled prediction to appended to the Elastic search, I appended them to the retrieved data. Then send back to the Elastic search and visualized them in Kibana dashboard.

In this test case, I wanted to make sure that the classification process runs with the ELK visualizing parallelly without getting any errors.

4.3 Implementation of Secure Blockchain

This section of the research focusses on storing the data out securely inside the blockchain and implementing blockchain on the cloud. To achieve these objective and sub objectives we have used mechanisms like Merkle Tree hashing, Proof of Work to ensure the security and the efficiency of the blockchain and implementing the blockchain in the cloud environment also protects consumers against several security issues. Moreover, access to encryption keys and unencrypted data for the end user may be provided only through Client-side encryption.

This process starts with the creating of the block and generating of the hash value. As you can see in the figure below, I have used some dummy data values to generate some block and their respective hashes. And at the same time the validation of the block also has being checked.

```
[Running] node "/Users/umindu/Desktop/main.js"
Mining block 1...
Block mined: 0000a2c165d52d5bb06d9a315acd15caba8a238d48a2156beb3b6055bfed2740
Mining block 2...
Block mined: 000016bf6368d4559e6b0753bb7c79d9ec4d8b8c1b3edbc3290650bf0d38eb8f
Mining block 3...
Block mined: 00003f04764436fc2f3601bcb1be1149c88931d59eae0400e7057ddcedb2d4fd
Mining block 4...
Block mined: 00006478859df1cd8af9445ac3d002ad8f6a1aecb80a5c3a7382b450618e4ae5
Mining block 5...
Block mined: 0000a483e45f45487493e04dcd2aa7d9a1990506f997c47039fb3a66e521eeb0
Is Blockchain Valid? true
Is Blockchain Valid? false

[Done] exited with code=0 in 2.084 seconds
```

Figure 49Figure 4.3.1 Implementation of Blocks and respective hashes

And then furthermore Merkle tree method is implemented to advance the security level of the blockchain. Following shown below is the Merkle tree hashing method.

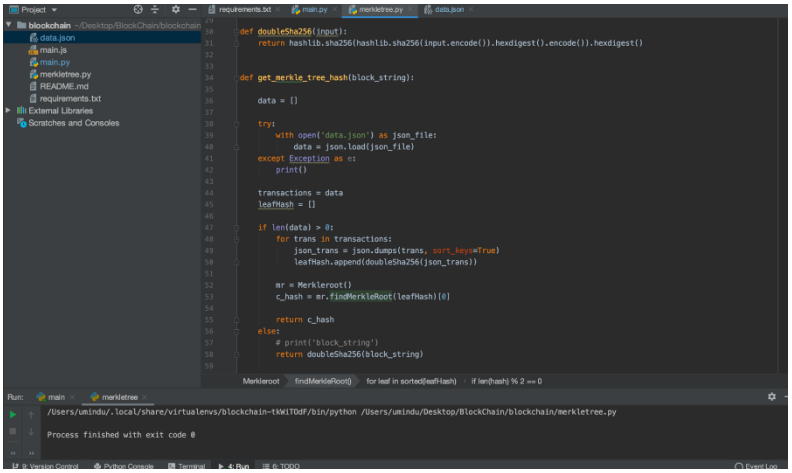


Figure 50Figure 4.3.2 Implementation of the Merkle root in Blockchai

Figure 51Figure 4.3.3 Test data output generated by Neuro Realistic Detection Engine

The *Figure* 4.3.3 are the sample of test data set that is used to check the storing of data in the. Blockchain from Neuro Realistic Detection Engine. (Refer Appendix I for detailed information)

To send the above data set to the blockchain POSTMAN is used so that the test dataset can be sent as a POST request so that it can be captured, hashed, and saved in the blockchain as a JSON file.

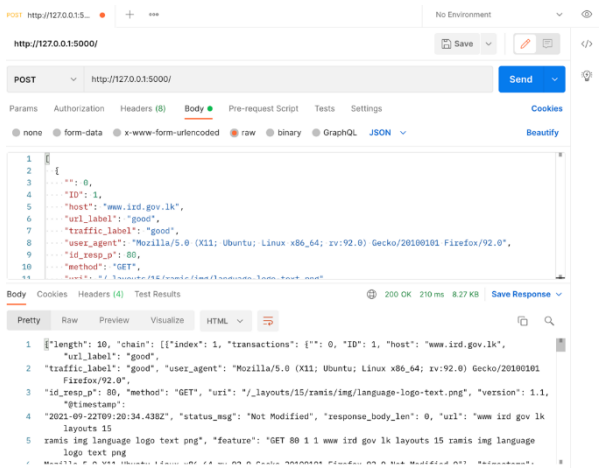


Figure 53igure 4.3.4 Response of Test data stored

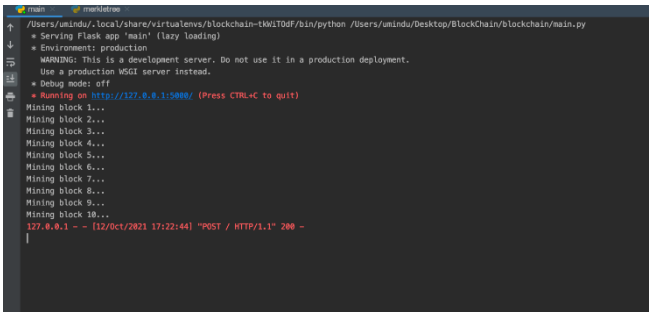


Figure 52Figure 4.3.5 Block Mining

According to the above *Figure* 4.3.5 *1*0 blocks have been mined because the test data output CSV file generated by the Neuro Realistic Detection Engine has 10 records which means all the data has successfully been saved inside the blockchain.

Also, the test dataset sent by the AI engine. (Refer Appendix II for detailed information) Data labels captured from the Artificial Engine is as follows.

5. RESULTS & DISCUSSION

5.1 Results

This part of the research which is the threat detection engine is responsible for capturing network data for anomaly detection. These data will be used by the AI engine and the neuro realistic engine to conduct further analysis on the data if it is malicious or not. As mentioned earlier, the threat detection engine is responsible to capture the features of the dataset of the AI engine and the neuro realistic engine.

According to the research the outputted data is visualized in the kibana dashboard, before displaying these results, it will be placed inside an index where we could find the captured features of the dataset.

In the process of passing data into elasticsearch, there is a middlemen called logstash who collects those data and do the filtering according to the desired output format. In this case grok patterns and KV pairs were used to split each column and its value to separate fields while displaying it in the kibana dashboard so that it would fit the script used by the AI engine and neuro realistic engine to capture those fields.

Figure 54Figure 5.1.1 result of field split using KV pairs

The above figure shows the results of the KV pair filter of logstash where the captured data has been split into each field.

5.1.1 Deep Autoencoder and Random Forest

After the preprocessing step of NSL-KDD dataset two different models were tested. One was a deep autoencoder with random forest and the other one was an artificial neural network. In the first method deep autoencoder was created with 123 input layer neurons because preprocessed features contained 123 dimensions. Apart from that there were four hidden layers and the bottleneck layer of the encoder part and four hidden layers and one output layer of the decoder part of the deep autoencoder. It was trained and validated with preprocessed features with 256 batch size and 50 epochs. The purpose of that was to obtain new meaningful reduced feature representation from the original features, generated utilizing the dimensional reduction mechanism from the encoder part of the deep autoencoder. Then the low dimensional features were given as the input for the random forest algorithm for further classification process.

	precision	recall	f1-score
access	0.91	0.86	0.88
dos	0.92	0.99	0.95
normal	0.99	0.98	0.98
privilege	0.50	0.14	0.22
probe	0.94	0.69	0.80

Figure 55Figure 5.1.1.1: performance evaluation of multiclass classification using deep autoencoder and random forest (NSLKDD)

```
Accuracy = 0.9784084143679301
Precision = 0.7486307389552902
Recall = 0.6063872594532211
```

Figure 56Figure 5.1.1.2: Overall performance evaluation of deep autoencoder and random forest model (NSL-KDD)

5.1.2 Artificial Neural Network (NSL-KDD)

Then the same dataset used to train with artificial neural network model which comprises one input layer two hidden layers and one output layer. This was trained and validated with preprocessed features with 256 batch size and 50 epochs. The

experimental results showed that the artificial neural network model has a stronger and higher detection rate than the hybrid model.

	precision	recall	f1-score
access	0.97	0.97	0.97
dos	1.00	1.00	1.00
normal	1.00	1.00	1.00
privilege	1.00	0.43	0.60
probe	1.00	1.00	1.00

Figure 57Figure 5.1.2.1: Performance evaluation of multiclass classification using artificial neural network (NSL-KDD)

```
Accuracy = 0.998571145068466
Precision = 0.9444260472398633
Recall = 0.8741266209024285
```

Figure 58Figure 5.1.2.2: Overall performance evaluation of artificial neural network (NSL-KDD)

5.1.3 Experimental results associated with CICDS dataset

The basic idea behind using NSL-KDD dataset was to have a comparison with former research studies and evaluate the progress of this research since all of them were used that as the benchmarking dataset and that was one of the problems identified during the literature survey as well. Hence. to overcome that problem as the next step of building AI engine, CICIDS dataset was used as the benchmarking dataset during this research.

5.1.4 Support Vector Machine

As shown in the image below SVM model was able to achieve 95.20 accuracy while classifying attacks and it took around 164 seconds to train and it was taken the second longest training time period out of all other machine learning algorithms. But when it comes to individual attack detections it was found out that SVM provided less detection accuracy for some attack categories like xss, bot, slowloris, bruteforce, sshpatator and more importantly it completely misclassified xss attack as bruteforce and benign.

```
Accuracy = 0.9520439398215488
Precision = 0.7669032541925443
Recall = 0.7347068276312194
```

Figure 59Figure 5.1.4.1: Overall performance evaluation of Support Vector Machine (CICIDS)

5.1.5 Decision Tree Classifier

Decision tree classifier was the next machine learning algorithm tested and while training phase it only took around 21 seconds and when compared with support vector machine algorithm this was quite low but and surprisingly it was able to achieve higher accuracy up to 99.77 as shown in the image below and it also provided less detection accuracy for some attack categories like xss, bot, and bruteforce attacks as shown in the confusion of the decision tree classifier.

```
Accuracy = 0.9977639320061654
Precision = 0.9307725256106912
Recall = 0.9303206721271288
```

Figure 60Figure 5.1.5.1: Overall performance evaluation of Decision Tree Classifier (CICIDS)

5.1.6 Naïve Bayes

Out of all the machine learning algorithms tested with CICIDS dataset this achieved the lowest accuracy and it was 82.58 and when compared with previous two it took longer time period than decision tree classifier to train but after observing the results of confusion matrix it was found out that this was classifying the fourteen attacks into nine categories.

```
Accuracy = 0.8285174652106897
Precision = 0.44587971678960275
Recall = 0.4494400991512876
```

Figure 61Figure 5.1.6.1: Overall performance evaluation of the Naïve Bayes (CICIDS)

5.1.7 Random Forest

When compared with all other machine learning algorithms used in this research the random forest had a higher accuracy than the others and it was 99.82. But when performing testing with new data it was found out that this model was misclassifying several attacks like xss and synflood due to the overfitting and dataset biased problems.

```
Accuracy = 0.9982343022961306
Precision = 0.9400667858815873
Recall = 0.9365330336722524
```

Figure 62Figure 5.1.7.1: Overall performance evaluation of the Random Forest (CICIDS)

5.1.8 Artificial Neural Network (CICIDS)

In the previous phase two deep learning models were tested and one was unsupervised deep auto encoder with random forest and the other one was supervised artificial neural network. There artificial neural network obtained more accuracy and due to that reason, it was decided to build artificial neural network using CICIDS dataset.

After several attempts the best output was achieved by setting up the ANN with one input layer, hidden layer, drop out layer and the output layer. There “Relu” was chosen as the activation function of the first hidden layer and “Softmax” was used as the activation function in the output layer. To be more precise, the output layer was set up with 14 neurons, one for representing each attack category in the classification process, Furthermore, “Softmax” activation function is used to normalize the outputs which enables transforming them from weighted sum values to probabilities that add to one. Each number in the “Softmax”'s output is interpreted as the likelihood of to each attack category also “Adam” optimizer used to leverage the learning rate of the model and sparse categorical cross entropy used as the loss function for training the proposed ANN.

In addition to that model was trained with 200 epochs and 512 batch sizes setting up the early stopping function specifying the loss function and it took around 585 seconds to fully train the model as shown in the image below.

```
Accuracy = 0.987864446518898
Precision = 0.8513068358005057
Recall = 0.9337619739431128
```

Figure 63Figure 5.1.8.1: Overall performance evaluation of the ANN (CICIDS)

col_0	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Label														
benign	44475	704	54	15	45	110	0	58	29	24	37	86	2	160
bot	2	381	0	0	0	0	0	0	0	0	0	0	0	0
bruteforce	1	0	81	0	0	0	0	0	0	0	0	44	0	168
ddos	14	0	0	25521	0	5	0	0	0	0	0	0	0	0
ftppatator	2	0	0	0	1536	0	0	0	0	4	0	0	0	0
goldeneye	5	0	0	0	0	5727	0	0	2	0	1	16	0	0
hulk	0	0	0	0	0	0	777	0	0	0	0	0	0	0
portscan	35	0	0	0	0	0	0	31656	0	5	0	0	0	0
slowhttptest	4	0	0	0	0	2	0	0	1075	7	0	1	0	1
slowloris	1	0	0	0	0	0	0	0	5	1153	0	0	0	0
sql	4	0	0	0	0	1	0	0	0	0	2368	0	0	0
sshpatator	0	0	0	0	1	0	0	0	0	2	0	1216	0	0
synflood	0	0	0	0	0	0	0	0	0	0	0	0	20418	0
xss	0	0	17	0	0	0	0	1	0	0	0	2	0	128

Figure 64Figure 5.1.8.2: Confusion matrix of the ANN (CICIDS)

```
print(classification_report(Labels_Test, labels_test_predictions
```

	precision	recall	f1-score	support
benign	1.00	0.97	0.98	45799
bot	0.35	0.99	0.52	383
bruteforce	0.53	0.28	0.36	294
ddos	1.00	1.00	1.00	25540
ftppatator	0.97	1.00	0.98	1542
goldeneye	0.98	1.00	0.99	5751
hulk	1.00	1.00	1.00	777
portscan	1.00	1.00	1.00	31696
slowhttptest	0.97	0.99	0.98	1090
slowloris	0.96	0.99	0.98	1159
sql	0.98	1.00	0.99	2373
sshpatator	0.89	1.00	0.94	1219
synflood	1.00	1.00	1.00	20418
xss	0.28	0.86	0.42	148
accuracy			0.99	138189
macro avg	0.85	0.93	0.87	138189
weighted avg	0.99	0.99	0.99	138189

Figure 65Figure 5.1.8.3: Classification report of the ANN (CICIDS)

This feed forward neural network obtained good results and outperformed all the other machine learning algorithms and eliminated overfitting deficiencies identified in the random forest model. Then it was tested against newly created attack samples according to the approach explained in the testing section and following images depicts the outcome of those. There it was found out that employed FFNN detects benign, ftp patator, goldeneye, hulk, portscan, slowhttptest, slowloris, sql, ssh patator and synflood attack categories with moderate accuracy of 97% and only drawback was less sensitivity against bruteforce attack.

5.2 Test case 01 results

In the test case 03 I trained the network classification model by merging both our and the HTTP CSIC datasets. This test case training and testing results fulfilled my expectations. The model training accuracy was 86% and the training loss was 0.23 (figure 6.1.5). Though it was not a higher accuracy, it was above the 80% under the standards and the testing results did not overfitted. As described on figure 6.1.6 in the testing it gave 0.85 accuracy with 0.81 precision for classifying abnormal traffics, 0.88 precision for normal traffics and 0.83 recall percentage for the abnormal traffics and 0.86 recall for the normal traffics.

```
loss: 0.2386 - accuracy: 0.8580

loss: 0.2385 - accuracy: 0.8602

loss: 0.2376 - accuracy: 0.8618

loss: 0.2416 - accuracy: 0.8629
```

Figure 66Figure 5.2.1: Test case 03 model training results

Since this model performed well I chose this model as the finalized network traffic classification model. Later on deployed the model on the Vertex AI to integrate with our final product.

```
Accuracy: 0.85
Detail:
              precision    recall  f1-score

         0.0       0.81      0.83      0.82
         1.0       0.88      0.86      0.87

    accuracy                           0.85
   macro avg       0.84      0.84      0.84
weighted avg       0.85      0.85      0.85
```

Figure 67Figure 5.2.2: Test case 03 model testing results

5.3 Test case 02 results

Train and test the malicious URLs classification model was the test case 04. As described in the test case 04, the model was trained using the merged dataset which included our dataset and the Malicious URL dataset. Here I was able to get 98% training accuracy with 0.05 loss (figure 6.1.7). The model testing results was 0.97 overall accuracy with 0.95 precision for classifying malicious URLs, 0.98 precision for normal URLs and 0.87 recall percentage for the malicious URLs and 0.99 recall for the normal URLs (figure 6.1.8).

```
loss: 0.0497 - accuracy: 0.9812

loss: 0.0501 - accuracy: 0.9810

loss: 0.0500 - accuracy: 0.9811

loss: 0.0496 - accuracy: 0.9813
```

Figure 68Figure 5.3.1: Test case 04 model training results

This model gave higher accuracy without overfitting, therefore, I decided to choose this model for the integration.

```
Accuracy: 0.97
Detail:
              precision    recall  f1-score

         0.0       0.95      0.87      0.91
         1.0       0.98      0.99      0.98

    accuracy                           0.97
   macro avg       0.96      0.93      0.95
weighted avg       0.97      0.97      0.97
```

Figure 69Figure 5.3.2: Test case 04 model testing results

5.4 Test case 03 results

I randomly selected 10 network traffic rows from our testing dataset and send JSON requests to the Vertex AI API for get prediction responses while measuring the time. I calculated the response time peak and off-peak hours, and the test case results are shown in the table 5. When taking average response time per day, it gave **. These results were calculated for both network traffic and URL classification model.

Table 6Table 5.4.1: Test case 05 models' response time

	Time	**Response time (seconds)**	**Average response time (seconds)**	**Average response time per day For 10 records (seconds)**	**Average responsetime per day For 1 record (seconds)**
Peak time	7.00 p.m.	10.8716	28.7876	9.5958	8.5985
	9.00 p.m.	9.8321			
	11.00 a.m.	8.0839			
Off-peak time	8.30 a.m.	6.2639	30.4051		
	12.00 p.m.	7.7068			
	2.30 p.m.	7.9709			

5.5 Test case 04 results

The detection system and the ELK were run parallelly without any interruptions and without consuming many resources in my local machine. Since the DL models, prediction get through the cloud, the detection system did not consume the infrastructure resources. After transferring prediction data into Kibana, they were visualized to end user using a dashboard as shown in the figure 6.1.9.

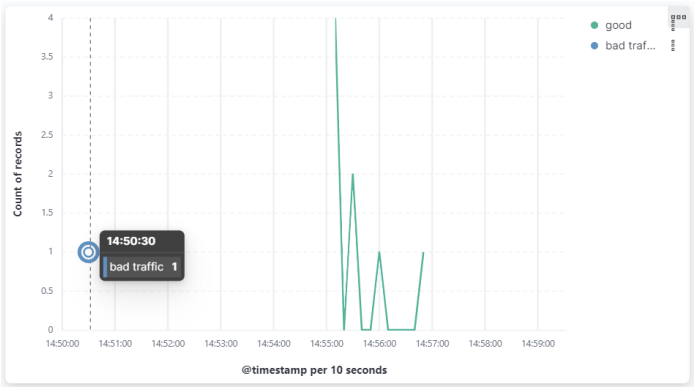


Figure 70Figure 5.5.1 Kibana dashboard which visualized prediction results

5.6 Results of blockchain

This part of the research shows all the result outputs that being stored inside the blockchain. These data that is stored inside the blockchain is used to as a report to get an understanding about all the network traffic that is passed through the autonomous cyber AI. As mentioned earlier the blockchain stores the data that is being transferred Neuro Realistic Detection Engine.

According to the research the outputted data from the Neuro realistic detection engine is stored in separate blocks. And is written into file in JSON format for later usage.

```
{
    "": 1,
    "ID": 2,
    "host": "www.ird.gov.lk",
    "url_label": "good",
    "traffic_label": "good",
    "user_agent": "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:92.0) Gecko/20100101 Firefox/92.0",
    "id_resp_p": 80,
    "method": "GET",
    "uri": "/_layouts/15/ramis/img/ta.png",
    "version": 1.1,
    "@timestamp": "2021-09-22T09:20:34.439Z",
    "status_msg": "Not Modified",
    "response_body_len": 0,
    "url": "www ird gov lk  layouts 15 ramis img ta png",
    "feature": "GET 80 1 1 www ird gov lk   layouts 15 ramis img ta png Mozilla 5 0  X11  Ubuntu  Linux x86 64  rv
},
{
    "": 2,
    "ID": 3,
    "host": "www.ird.gov.lk",
    "url_label": "good",
    "traffic_label": "good",
    "user_agent": "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:92.0) Gecko/20100101 Firefox/92.0",
    "id_resp_p": 80,
    "method": "GET",
    "uri": "/_layouts/15/ramis/img/shadow.png",
    "version": 1.1,
    "@timestamp": "2021-09-22T09:20:34.532Z",
```


Figure 71Figure 5.6.1 Test data of Neuro realistic detection engine stored in Blockchain in JSON format

Figure 5.6.1 shows the CSV generated by the Neuro Realistic Detection Engine is successfully stored securely in the blockchain in JSON format.

```
{
    "src_ip": "192.168.85.137",
    "dst_ip": "13.225.0.16",
    "dst_port": 443,
    "flow_duration": 10240070,
    "tot_fwd_pkts": 2,
    "tot_bwd_pkts": 2,
    "totlen_fwd_pkts": 108,
    "totlen_bwd_pkts": 120,
    "fwd_pkt_len_max": 54,
    "fwd_pkt_len_min": 54,
    "fwd_pkt_len_mean": 54,
    "fwd_pkt_len_std": 0,
    "bwd_pkt_len_max": 60,
    "bwd_pkt_len_min": 60,
    "bwd_pkt_len_mean": 60,
    "bwd_pkt_len_std": 0,
    "flow_byts_s": 22.26547279,
    "flow_pkts_s": 0.39062233,
    "flow_iat_mean": 3413356.667,
    "flow_iat_std": 4826995.381,
    "flow_iat_max": 10239759,
    "flow_iat_min": 142,
    "fwd_iat_tot": 0,
    "fwd_iat_mean": 0,
    "fwd_iat_std": 0,
    "fwd_iat_max": 0,
    "fwd_iat_min": 0,
    "bwd_iat_tot": 0,
    "bwd_iat_mean": 0,
    "bwd_iat_std": 0,
    "bwd_iat_max": 0,
    "bwd_iat_min": 0,
    "fwd_psh_flags": 0,
    "bwd_psh_flags": 0,
    "fwd_urg_flags": 0,
    "bwd_urg_flags": 0,
    "fwd_header_len": 40,
    "bwd_header_len": 40,
    "fwd_pkts_s": 0.195311165,
    "bwd_pkts_s": 0.195311165,
    "pkt_len_min": 54,
    "pkt_len_max": 60,
    "pkt_len_mean": 57,
    "pkt_len_std": 3,
    "pkt_len_var": 9,
    "fin_flag_cnt": 1,
    "syn_flag_cnt": 0,
```


Figure 72Figure 5.6.2 Test data of AI engine stored in Blockchain in JSON format

6. Research Findings

With the conducted research, we were able to realize that there we no proper open-source SIEMS based on anomaly detection. Most of the legacy systems use signature based or predefined rules in order detect malicious activities in a system. Therefore, there was a need in the market for a system to be built to detect anomaly activities including zero days using artificial intelligence mechanisms. And there were no datasets which has been created for particular attacks so that we have to simulate some attacks to create datasets in order to test the system. While capturing data, it was observed that packetbeat only is not enough in order to capture the features of the network traffic required by the Artificial Intelligence mechanisms of the system to conduct the threat detection. Therefore, it was needed to bring in zeek and CICFlowmeter which goes along with the system and helps to capture the required fields of data which then is integrated with beats and those data are passed to logstash. Once the CICflowmeter was integrated into the system, we were able to identify that it is capable to extract 100% features of the CICIDS2017 dataset which provided a huge boost in the research. And also using logstash filter techniques such as KV pairs and grok pattern it was able find out how to split data to each field. The filtering techniques were very useful in the system integration process where a python script was executed and the key features in the script was the separated fields done by the logstash filtering. And also, was proposed to build a mechanism solely utilizing unsupervised deep learning approach using deep stack autoencoders and as expected a deep stack autoencoder model was successfully implemented using NSL-KDD dataset. After thorough study conducted about the results it was found out that the model was tends to misclassify attack instances as normal and that was a crucial problem to the reliability and the accuracy of the AI engine. Therefore, it was then used as a feature extraction tool utilizing the dimensional reduction mechanism of the encoder part. There the outcome of the encoder model was given as the input for random forest algorithm, and it was able to obtain successful results when compared to the previous model. Since the ultimate outcome was supervised based, it made more sense than the output of the unsupervised approach. Thus, it was decided to take supervised approach into consideration to build the final model and FFNN was also

built using the same NSL-KDD dataset. There it outperformed than the hybrid model and it was used to compare the machine learning approaches with deep learning while using the CICIDS dataset. Out of all the supervised machine learning algorithms random forest obtained the highest accuracy but surprisingly after exposing the trained model to the new network traffic instances it completely misclassified them and found out it has overfitting problems and dataset bias issues. Then FFNN was implemented using CICIDS and after using the trained model to classify new network traffic instances it achieved the promising results with high accuracy. Therefore it is proven the fact that deep learning models always perform better than machine learning approaches. And also, Through this NLP based network intrusion detection system implementation research components, there are specific research findings to consider. From the model training and testing phases where I discussed above, my model training expectation was getting more than 80% accuracy with lower model loss. As I mentioned earlier, from the first test case scenario I was able to get only 66% accuracy for the network traffic classification model with a higher loss. In that test case I only used the HTTP CSIC dataset. Since it was a low-quality model which have lower accuracy, I did not choose the model for the integration. After I trained and tested the model using over dataset as described in the second test case. Here I was able to get 99% accuracy. It was beyond my expected outcome. Therefore, I further tested that model with manually labeled dataset and I realized that the model was overfitted. Though the model had a higher accuracy, it was not a good decision to choose overfitted model for the integration. After I found the model overfitted reason was data balancing of our training dataset, I merged it with the HTTP CSIC dataset and retested the model as described in third test case. From that, the model got 85% accuracy. Although that accuracy not quite enough for an IDS, it was a good achievement that reaching 85% accuracy form 66% without overfitting the model. Therefore, I decided to select this model as my finalized network traffic classification model and used it for the integrations. Third test case for the malicious URLs classification model. Here I got 97% good accuracy by training merged dataset that made with Malicious URLs dataset and our dataset. After deploying both trained models on Vertex AI, I analyzed the response time for get predictions where is described in the fifth test case. Model response time was main part of intrusion

detection system because user notification completely depends on model response time. According to the results, average peak hours both models' response time was 28.7876 seconds and the off-peak hours it was 30.4051. That means the models take averagely 19.7309 seconds for give prediction response within day. Final test case was testing whole process of the neuro realistic detection system from real time data retrieving to real time data visualizing. The detection system should run with the ELK stack parallelly without any interruptions. It performed well in the testing. According to those findings neuro realistic detection system was implemented more than expected to implement. With this research conducted, we were able to realize that there we no proper open-source SIEMS based on anomaly detection and no other SIEM used blockchain to store the generated data. Most of the legacy systems use signature based or predefined rules in order detect malicious activities in a system and to store the data generated thy use cloud storage. Therefore, there was a need in the market for a system to be built to detect anomaly activities including zero days using artificial intelligence mechanisms and a more secure wat to store the generated results While designing the blockchain it was observed that just hashing and proof of work is insufficient to maintain higher level of security for the stored data inside the blockchain which is generated from the Neuro Realistic Detection System and the AI engine. So Merkle tree hashing was introduced which enables double hashing and it will Therefore, it dramatically lowers hash levels, allowing for quicker verification and transactions. And implanting the blockchain in the cloud environment users are linked with a safe P2P network in distributed cloud storage that provides decentralized storage without affecting safety on vital data for their assignment. Blockchain technology offers more powerful information retention characteristics. The cost may be cut to half by ten times faster. The strategic mix of distributed cloud storage encompasses openness, safety and hash features, private/public encryption, and secure transaction leads. Decentralized cloud storage also protects consumers against several security issues. Moreover, access to encryption keys and unencrypted data for the end user may be provided only through Client-side encryption. All this enables clients to fully manage their different data assets.

6.1 Discussion

Capturing of network-based and host-based data were the main components of the threat intrusion detection engine. Once implemented these components it was a mandatory task to test and check if the result of the component is matching with expected outcome. There were two main datasets in the autonomous cyber ai which is used by the ML models where those features of the datasets needed to be captured. Firstly, we began using packetbeat which gave us about 30% of the result we wanted and we wanted better was and mechanisms to capture all features. Eventually we came across CICFlowmeter which was able to provide 90% of the results we wanted and gave the system a boost. The main discussion here is what more has to be improved in this data capturing process? According to the research it was a bit complex scenario running many servers and services at the same time. In would indeed be more convenient and also provide a more efficient and effective mechanism if we could integrate all services into one and run just one script to start all servers and services.

This research was conducted under two main subcomponents, under one subcomponent ML and DL models were implemented and evaluated the performance against former research studies that had been identified during the literature survey and other one was the new approach taken to fill out the drawbacks identified in existing approaches. In the first phase experiment was done using a well-known dataset called NSL-KDD which was widely used in security domain over several decades to build different intrusion detection models. There two main approaches were taken, and one was supervised deep learning model with FFNN and other one was a hybrid model which comprised with autoencoders which is an unsupervised deep learning algorithm with random forest machine learning model. After experimenting several times best accuracy was achieved using FFNN and that was 99.85. Then former studies were compared with our model to verify the overall model performance. There, Yin et al [7] study used Recurrent Neural Network for network traffic classification, and they obtained 99.81 accuracy and authors of the Shone et al [8] proved that their Non-symmetric Deep Autoencoder model was able to achieve 97.85 accuracy level. In another research study called as Ali et al [9] utilized ANN for classification purpose and authors claimed that the model obtained accuracy of 99.69 in the final evaluation

process. Another study called Jia et al [10] claimed that they were able to obtain 98.99 accuracy while Al-Qatf et al [11] study proposed a method which was implemented based on autoencoders and SVM and only able to achieve the accuracy of 93.96. Moreover, all of those former research studies utilized NSL-KDD dataset and there we were able to prove that our approach which is based on FFNN obtained promising results than them. In the next phase CICIDS used as the benchmarking dataset and tested with several machine learning models and FFNN. Here also FFNN was identified as the best model which obtained the accuracy of 98.78 and chosen as the feasible model for building the proposed AI engine.

Network Traffic Classification model and Malicious URLs Classification model was the main components of the Neuro Realistic Detection System. After implementing those main components, it was compulsory to test and check those implementations are working properly and those implementations provide expected outcomes. There were six test cases to identify DL models and the integration to the ELK implemented as expected. From first four test cases which were models training and evaluating main purpose was to identifying the models are performing properly. Training accuracy, loss and the testing accuracy were the main functions to identify about the models training process. In the first testing scenarios, the model only trained using a one dataset. From HTTP CSIS dataset, Malicious URL and the dataset which created by our team. For the training the datasets were split into two parts as train data and the test data. Model train accuracy and the loss were measured when the training using the train data. After training, test data was inputted to the model for get prediction. Then using those predicted outputs and the real outputs which were in the dataset, measured the trained models' overall accuracy with precision and recall. Next, I trained the models by merging selected datasets with ours. After done all those test cases, I realized that training model using combination of both selected and our dataset give more accuracy than the expected outcomes. After deploying trained models as API on the Vertex AI, my main purpose was to check the response time from cloud-based DL models, reason for that was when real time network traffic data send to cloud and get response through the cloud so there can be latency because of connectivity. Therefore, as my fifth test case, I calculated the response time in randomly selected peak hours and off-peak hours by sending API requests and measured the average response time

for the day. Expected response time was less than 10 seconds and the results gave good response time. Finally, I integrate the detection system with ELK and tested the whole system performance with real time data. By achieving my expected goals, the system run without any interruptions and errors.

Having to the store data generated by the AI engine and the Neuro Realistic Detection Engine securely and effectively were the main components of this section. After implementing these functions successfully testing for desired results and matching is with the desired outcome was a mandatory task.

Furthermore, the main two security mechanism used inside the blockchain are hashing and proof of work. Here in this implementation Pow was used to ensure the integrity of the blockchain and instead of hashing Merkle tree method was used, in a blockchain, Merkle trees assist remove duplication or erroneous transaction records. For instance, if someone tries to hack an entry on the blockchain and it seems as if it is more bitcoin than it really is, their fake entry does not match the rest of the hazards in a Merkle tree. This is because no transaction can be utilized by itself when checking transactions on a blockchain. With every other hash, all hash should be checked to make sure that nothing is altered or manipulated.

And the implantation of the blockchain is connected to a secure P2P network in cloud storage which allows decentralized storage without impacting security of important data. Technology Blockchain enables more powerful retention of information. Costs can be reduced by 10 to half. The strategic combination of distributed cloud storage includes openness, safety and hash capabilities, private/public coding, and safe transaction guidelines. Also, this protects the user from several security issues

6.2 Summary of Each Student's contribution

Name	IT Number	Topic	Contribution
Hussain M.H	IT18166934	Threat Detection Engine	Capturing Real time network based data for anomaly detection
De Silva H.W.D.T	IT18361728	AI Engine	Analyzing the captured data to find anomalies using deep learning models
Madhuwantha K.A.N	IT18175530	Neuro Realistic Engine	Analyzing the captured data to find anomalies using url detection with NLP
Liyanage U.I.D	IT18036626	Blockchain model	Implementing a cloud based blockchain to store the generated output securely and efficiently.

7. Conclusion

As explained NIDS act prominent role as the second line of defense in network security. In this research it is focused to implement an open-source software solution "Open Architecture Based Autonomous Cyber AI" which addresses and eliminates existing drawbacks in current NIDS arena such as lack of proper mechanisms to enrich captured raw network data, highly depending nature of the industry leading products like Snort, Suricata on signature databases which do not get updated frequently, requirement of lot of computational power to get backed by deep learning models and high risk available for data at rest due to the deficiencies of existing blockchain based solutions. The proposed system comprises of threat intelligence engine for capturing and enriching raw data, deep learning enabled AI engine for detecting network anomalies, neuro realistic anomaly detection system for malicious URL detection and blockchain based mechanism for storing information in more efficient, organized, and secured manner. There the threat intelligence engine captures the real time traffic, data from multiple sources and then process the captured data into a meaningful format by enriching and enhancing the captured information for further analysis, while identifying legitimate traffic and discard them from further analysis. Once relevant data has been extracted from the captured traffic, it will be passed to the AI engine and neuro realistic detection system to conduct further analysis. Ai engine identifies the anomalies of the network traffic in layer three and layer four in the ISO-OSI model and Neuro realistic detection system operates in the application layer which classifies http traffic data and the URLs separately by using two separate deep learning models. Furthermore, algorithms such as Support Vector Machine, Decision Tree, Naïve Bayes, Random Forest, Word2vec, Convolution Neural Network (CNN), Feed Forward Neural networks (FFNN), and Autoencoders are used during this research and after conducting thorough analysis FFNN, Word2vec and CNN was utilized to build aforementioned detection models and, NSL-KDD, CICIDS HTTP DATASET, CSIC 2010 used in parallel with the above algorithms in order to obtain high accuracy in threat detection. There the Secure blockchain model was used to store all the labeled information in an organized and secure manner. Apart from those experimental results proved that the new approach taken to implement the "Autonomous Cyber AI" fill out

the drawbacks identified in existing industry products and also capable of provisioning robust protection against existing computer networks with high accuracy, reliability, and cost-effective manner. In the final stage of this research study the all the systems were integrated with Elasticsearch and Kibana and delivered as a customer friendly product.

8. Reference list

[1] S. B. ALIAS, S. MANICKAM, AND M. M. KADHUM, “A STUDY ON PACKET CAPTURE MECHANISMS IN REAL TIME NETWORK TRAFFIC,” PROC. - 2013 INT. CONF. ADV. COMPUT. SCI. APPL. TECHNOL. ACSAT 2013, PP. 456–460, 2013, DOI: 10.1109/ACSAT.2013.95.

[2] V. LABAYEN, E. MAGAÑA, D. MORATÓ, AND M. IZAL, “ONLINE CLASSIFICATION OF USER ACTIVITIES USING MACHINE LEARNING ON NETWORK TRAFFIC,” COMPUT. NETWORKS, VOL. 181, NO. FEBRUARY, 2020, DOI: 10.1016/J.COMNET.2020.107557.

[3] Yin C, Zhu Y, Fei J, He X.; “A Deep Learning Approach for Intrusion Detection Using Recurrent Neural Networks”; IEEE

Access. 2017; 5:21954-21961. https://doi.org/10.1109/ACCESS.2017.2762418

[4] Shone N, Ngoc TN, Phai VD, Shi Q; “A deep learning approach to network intrusion detection” IEEE Trans Emerg Top

Comput Intell. 2018;2(1):41-50. https://doi.org/10.1109/TETCI.2017.2772792.

[5] Ali MH, Al Mohammed BAD, Ismail A, Zolkipli MF; “A new intrusion detection system based on fast learning network and particle swarm optimization”; IEEE Access. 2018;6:20255-20261. https://doi.org/10.1109/ACCESS.2018.2820092.

[6] Jia Y, Wang M, Wang Y. Network intrusion detection algorithm based on deep neural network. IET Inf Secur. 2018;13(1):48-53. https:// doi.org/10.1049/iet-ifs.2018.5258.

[8] Ferrag, Mohamed Amine, Maglaras, Leandros Moschoyiannis, Sotiris Janicke, Helge; “Deep learning for cyber security

intrusion detection: Approaches, datasets, and comparative study” ; Journal of Information Security and Applications

[9] Daniel Berman; The Complete Guide to the ELK Stack; Available: https://logz.io/learn/complete-guide-elk-stack/#intro

[10] M. Mimura and H. Tanaka, Reading network packets as a natural language for intrusion detection, vol. 10779 LNCS. Springer International Publishing, 2018.

[11] M. Mimura and H. Tanaka, Reading network packets as a natural language for intrusion detection, vol. 10779 LNCS. Springer International Publishing, 2018.

[12] J. Li, H. Zhang, and Z. Wei, "The Weighted Word2vec Paragraph Vectors for Anomaly Detection over HTTP Traffic," IEEE Access, vol. 8, pp. 141787–141798, 2020, doi: 10.1109/ACCESS.2020.3013849.

[13] K. Barot, J. Zhang, and S. W. Son, "Using Natural Language Processing Models for Understanding Network Anomalies," IEEE-hpec ,pp. 1–7, 2016.

[14]TensorFlow. 2021. Models & datasets | TensorFlow. [online] Available at: https://www.tensorflow.org/resources/modelsdatasets

[15] S. J. Stolfo, S. Hershkop, L. H. Bui, R. Ferster, and K. Wang, "Anomaly detection in computer security and an application to file system accesses," Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 3488 LNAI, no. May 2005, pp. 14–28, 2005, doi: 10.1007/11425274_2.

[16] HTTP DATASET CSIC 2010 [Online]. Available: http://www.isi.csic.es/dataset

[17] GitHub. 2021. GitHub - faizann24/Using-machine-learningto-detect-malicious-URLs: Machine Learning and Security | Using machine learning to detect malicious URLs. [online] Available at: https://github.com/faizann24/Using-machinelearning-to-detect-malicious-URLs

[18]Srishty Choudhary, Uday Patkar; DATABASES FOR ARTIFICIAL INTELLIGENCE; Srishty Choudhary et al, International Journal of Computer Science and Mobile Computing, Vol.5 Issue.3, March- 2016, pg. 67-70

[19]Ms. Shweta Balasaheb Mirajkar, Prof. S. D. Khatawkar; A Provenance- Based Access Control Model for Securely Storing Data in Cloud; 2017 2nd International Conference for Convergence in Technology (I2CT)

[20]Saqib Ali†, Guojun Wang, Bebo White, Roger Leslie Cottrell; A Blockchain-based Decentralized Data Storage and Access Framework for PingER; 2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And

Communications/ 12th IEEE International Conference On Big Data Science And Engineering

[21]WEIZHI MENG 1 , (Member, IEEE), ELMAR WOLFGANG TISCHHAUSER1 , QINGJU WANG1 , YU WANG 2 , AND JINGUANG HAN3 , (Member, IEEE); When Intrusion Detection Meets Blockchain Technology: A Review