International Baccalaureate

Extended Essay

Computer Science

Investigating accuracy of Machine Learning in detecting HTTP DoS attacks

Research Question:

To what degree of accuracy is machine learning effective in detecting http flood DoS attacks on systems?

contact: research[at]aryanmukherjee.com

Contents

List of Tables

List of Figures

List of Algorithms

1 Introduction

In the rapidly evolving digital landscape, cybersecurity has become a topic of paramount importance. Among the plethora of cyber threats, Denial of Service (DoS) attacks pose a uniquely disruptive challenge. These attacks usually inundate targeted systems with an overwhelming number of requests, consequently disabling them and denying access to legitimate users, hence the name 'Denial of service'. Specifically, HTTP flood DoS attacks have been recognized for their simplicity to set up and potential for widespread disruption, as evidenced by their deployment in the recent Russia-Ukraine cyberwar, where they targeted critical infrastructure with significant impact (Kirichenko).

Given the severe consequences associated with these attacks, devising robust detection mechanisms are of utmost importance. This research proposes an unorthodox approach to detect HTTP flood DoS attacks by leveraging the power of machine learning. More specifically, a neural network model is trained and tested to detect these disruptive cyber threats.

This research examines the accuracy of binary classifiers, based on neural networks, in accurately detecting denial of service (DoS) attacks. DoS attacks come in many forms but this research intends to focus on HTTP flood attacks as they are the easiest to replicate by a cyber miscreant and therefore, the most common.

1.1 Motivation

As we navigate an increasingly digital era where the sheer volume of internet-connected devices increases at a steady rate, cyberattacks are becoming more and more commonplace (Abuse of vulnerabilities). As such, the imperative for autonomous threat detection systems is more pronounced than ever, particularly in light of the progressively sophisticated and evolving strategies adopted by threat actors (Delplace et al.).

The impetus for this research lies in my personal concern over the surge in DDoS attacks targeting critical infrastructure, as seen during the conflict in Ukraine. While it would be theoretically possible to replicate such a DDoS attack, doing so would involve illegitimate and unethical methods beyond the purview of this extended essay. Thus, I have chosen to simulate a more feasible DoS attack within the confines of my own systems, to ensure the legal and ethical integrity of this research endeavour.

Therefore, in this study, I had endeavoured to construct a neural network-based machine learning model to detect HTTP flood attacks. This autonomous model will facilitate the implementation of zero trust policies and eliminate human bias from the threat detection process.

1.2 Research Question

For the reasons outlined above, this report pursues the research question: **To what degree of accuracy is machine learning effective in detecting HTTP flood DoS attacks on systems?**

1.3 Aim

The primary aim of this extended essay is to explore and evaluate the efficacy of machine learning, specifically neural network based binary classifiers, in detecting HTTP Flood DoS attacks. This study seeks to understand the level of accuracy and reliability of Neural Networks which can achieve in distinguishing between benign and malicious network traffic patterns.

2 Literature Review

After the development of the aim, a literature review was conducted to gain a better understanding of the research question and its feasibility.

2.1 Machine Learning and Neural Networks

Machine learning is a technique that teaches computers to 'learn' from experiences. Machine learning algorithms use computational methods to 'learn' information directly from data without relying on a predetermined equation as a model. The algorithms improve their performance in an adaptive manner as the number of samples available for learning increases (Works). 'A neural network is a specific type of machine learning model that teaches a computer program to learn from data, based very loosely on how the human brain works. First, a collection of software "neurons" are created and connected together, allowing them to send messages to each other. Next, the network is asked to solve a problem, which it attempts to do over and over, each time strengthening the connections that lead to success and diminishing those that lead to failure' (Smilkov and Carter).

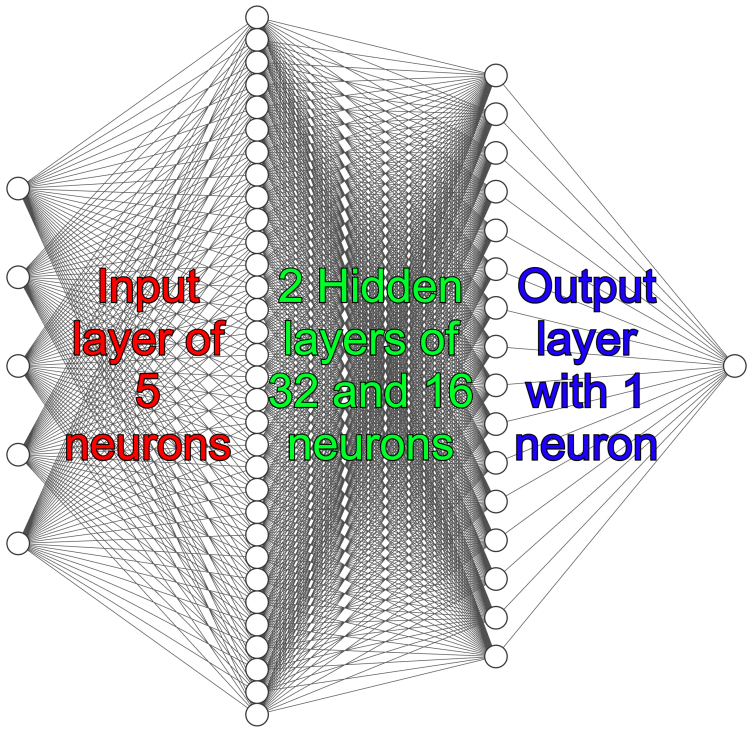

Figure 1: Visualization of the Neural Network used for the experiment (Alexlenail)

The above figure, represents the neural network that was used for this research. The type of Neural Network used is referred to as a 'Dense' Neural Network, which means that every neuron in a layer is connected to every neuron in the next layer, except for the final output neuron.

Therefore, through repeated manipulation of biases between neurons, a Neural Network can successfully classify network traffic into malicious and benign. However, the neural network needs data to train and set biases, which leads into the next topic.

2.2 HTTP requests and HTTP flood

An HTTP request is a request made by a client to a named host, like the type of request a web browser makes when visiting a website. An HTTP flood attack is a type of Denial-of-Service (DoS) attack aimed at incapacitating the targetted server with HTTP requests. Once the target has been saturated with requests, it is unable to respond to traffic from normal users (Cloudflare). As this was central to the crux of the research question, data on these attacks needed to be collected.

2.3 Wireshark

To detect and capture the network traffic data, a network sniffing tool called 'Wireshark' was used. 'Wireshark® is a network protocol analyser. Allowing the user to capture and view network traffic' (Wireshark). This tool was used to capture the network traffic for the experiment. The network traffic extracted from the data is usually stored in a file format called '.pcap' or '.pcapng' but was converted into '.csv' files for ease of use.

2.4 Graphs and Metrics

After the collection, training and testing of the neural network, it needs to be evaluated using certain metrics like accuracy, which can help paint a fuller picture of the performance of a neural network for a given task.

In particular, the confusion matrix, receiver operating characteristic (ROC) curve, area under the curve (AUC) and accuracy, are important metrics used in machine learning and therefore utilized in this extended essay.

2.5 Confusion Matrix

'By definition a confusion matrix C is such that $C_{i,j}$ is equal to the number of observations known to be in group i and predicted to be in group j '(Confusion Matrix). In this case, the machine learning model used is a binary classifier, or in other words, a classifier with only 2 outputs (1 and 0). 1 represents a malicious packet and 0 represents a benign packet. Thus, a 2×2 matrix is used and 'the counts of true negatives are $C_{0,0}$, false negatives are $C_{1,0}$, false positives is $C_{0,1}$, and true positives is $C_{1,1}$' (Confusion Matrix). Ideally, a perfect classifier should have a value of 0 for $C_{1,0}$ and $C_{1,0}$ (lower is better), and the sum of $C_{0,0}$ and $C_{1,1}$ (higher is better), should be equal to the total number of data points.

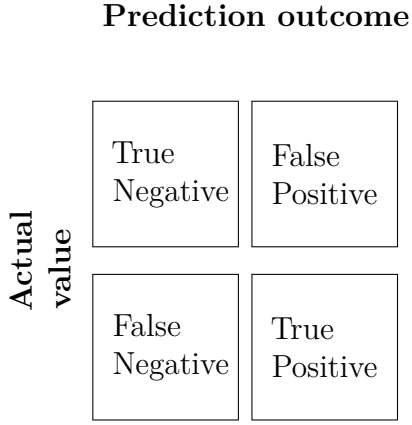

Figure 2: Representation of a 2 x 2 Confusion Matrix (Medina)

2.6 Reciver Operature Characteristics (ROC) Curve

An ROC curve is a graph showing the performance of a classification model at all classification thresholds. This curve plots two parameters: True Positive Rate (TPR) and False Positive Rate (FPR). Therefore, TRP is defined as:

$$TPR = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

Figure 3: Formula for True Positive Rate (Google)

False Positive Rate (FPR) is defined as follows:

$$FPR = \frac{False\ Positives}{True\ Positives + False\ Negatives}$$

Figure 4: Formula for False Positive Rate (Google)

An ROC curve plots TPR vs. FPR at different classification thresholds. Lowering the classification threshold classifies more items as positive, thus increasing both False Positives and True Positives. (Google)

The (ROC) curve also provides valuable insights into the performance of a model, showcasing the trade-off between its true positive rate and false positive rate across different classification thresholds.

2.7 Area under the curve (AUC)

The AUC metric complements the ROC curve. Hence, AUC stands for 'Area under the (ROC) curve.' That is, AUC measures the entire two-dimensional area underneath the

entire ROC curve (integral of the ROC curve) from (0,0) to (1,1). AUC provides an aggregate measure of performance across all possible classification thresholds. One way of interpreting AUC is as the probability that the model ranks a random positive example more highly than a random negative example (Google).

2.8 Accuracy

Accuracy is a metric that generally describes how the model performs across all classes. It is useful when all classes are of equal importance. It is calculated as the ratio between the number of correct predictions to the total number of predictions. The higher the value of accuracy, the better the model performs. This metric is what this research aims to focus on.

$$Accuracy = \frac{Total\ correct\ predictions}{Total\ predictions}$$

Figure 5: Formula for accuracy (Gad)

2.9 Validation accuracy and Accuracy

When training a machine learning model, one of the main things that is desired is to avoid, would be overfitting. This is when the model fits the training data well, but it is not able to generalize and make accurate predictions for data it has not seen before. To find out if their model is overfitting, data scientists use a technique called cross-validation, where they split their data into two parts - the training set, and the validation set. The training set is used to train the model, while the validation set is only used to evaluate the model's performance. Metrics on the training set let you see how the model is progressing in terms of its training, but it's metrics on the validation set that let you get a measure of the quality of your model - how well it is able to make new predictions based on data it hasn't seen before (Primusa and Traina). Therefore, accuracy represents the accuracy derived from the training dataset and validation accuracy is the accuracy when predicting the labels for the validation dataset.

3 Methodology

3.1 Data Collection

As stated in Machine Learning and Neural Networks (section 2.1), the dataset on which the Neural Network is trained, plays an important role in the performance of the Neural Network. Hence, to train the Neural Network, an HTTP DoS attack was simulated in a controlled environment. An open-source penetration testing tool called 'PyFlooder' (PyFlooder) was used. It was then configured to request the main page of a webserver hosted on a local server. The webserver was a simple web page, hosted using a python

framework called Flask. The server was also running Wireshark to capture the incoming network traffic from the attack.

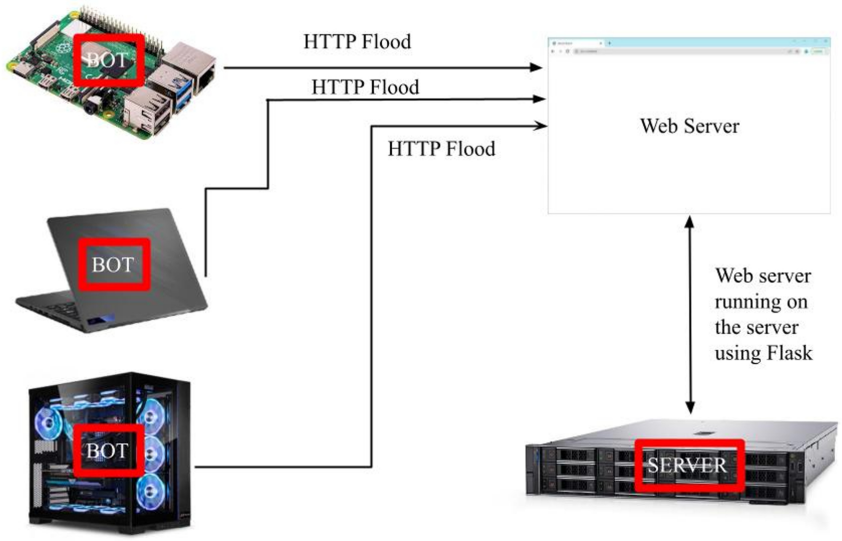

Figure 6: Setup of experiment

While the network traffic representing HTTP flood was gathered, to achieve a reasonable level of accuracy, the Neural Network also needs to be trained with benign data. Without this procedure, it is possible that the Neural Network interprets every network traffic as malicious, thereby rendering it ineffective. To combat this issue, network traffic was also collected from the day-to-day use of a computer.

3.2 Data

After the data from the two experiments was collected, it was converted into '.csv' files to allow for easier interfacing using a python library called Pandas. This library is aimed at managing datasets. After this, the data was labelled, based on which experiment it was derived from, into two categories: malicious and benign. The final dataset, had a count of 132,550 packets. This included 66,937 malicious packets and 65,613 benign ones, highlighting the balance of the dataset used in this research to minimize any bias that might arise.

The consolidated dataset is displayed below, featuring all its attributes such as timestamp, source IP, destination IP, protocol, length, and the label assigned. While the original network traffic had more attributes, they were stripped as most of them do not share a strong correlation to whether or not the packet is malicious, using the filter method, which 'as the name suggests, involves filtering features in order to select the best subset of features for training the model' (see 'Filter method' (Delplace et al.)).

Network Data						
#	timestamp	source IP	destination IP	protocol	length	label
0	00073.92982	[REDACTED]	[REDACTED]	HTTP	448	malicious
1	00074.68644	[REDACTED]	[REDACTED]	HTTP	468	malicious
2	00076.94549	[REDACTED]	[REDACTED]	HTTP	480	malicious
3	00084.26156	[REDACTED]	[REDACTED]	HTTP	572	malicious
4	00084.86820	[REDACTED]	[REDACTED]	HTTP	632	malicious
...	...	...	...	...	...	...
132546	76350.84727	[REDACTED]	[REDACTED]	DNS	64	benign
132547	76350.84747	[REDACTED]	[REDACTED]	DNS	59	benign
132548	76350.84851	[REDACTED]	[REDACTED]	DNS	64	benign
132549	76350.85364	[REDACTED]	[REDACTED]	DNS	59	benign
132550	76350.85476	[REDACTED]	[REDACTED]	DNS	64	benign

Table 1: Raw Network Data

The timestamp represented the time each packet was transmitted, while the source and destination IPs denoted the origin and recipient of each packet, respectively. The protocol attribute identified the communication protocol used by the packet. The length attribute indicated the size of each packet, measured in bytes. The label attribute played a pivotal role in differentiating between benign and malicious packets. This differentiation was facilitated by a Python script, which matched the IP to a list of malicious IPs predetermined through the static IP configuration of the systems running malicious scripts in the experiment. As stated in the previous section, this was also cross-validated by checking which of the two experiments, the data was derrived from.

The IP addresses where redacted as they were public IP addresses. It was hypothesised that if private IPs were used, the machine learning model could unfairly bias against internal traffic (as private IPs are designed for internal or private use), which was unsuitable.

3.3 Feature Extraction

After the raw data was collected, the timestamp column, which represents the time at which each packet was transmitted, was replaced by time delta, which was computing the time difference between consecutive packets sharing the same source IP. This was accomplished by first sorting the data by its IP, and then calculating the time delta between successive data points.

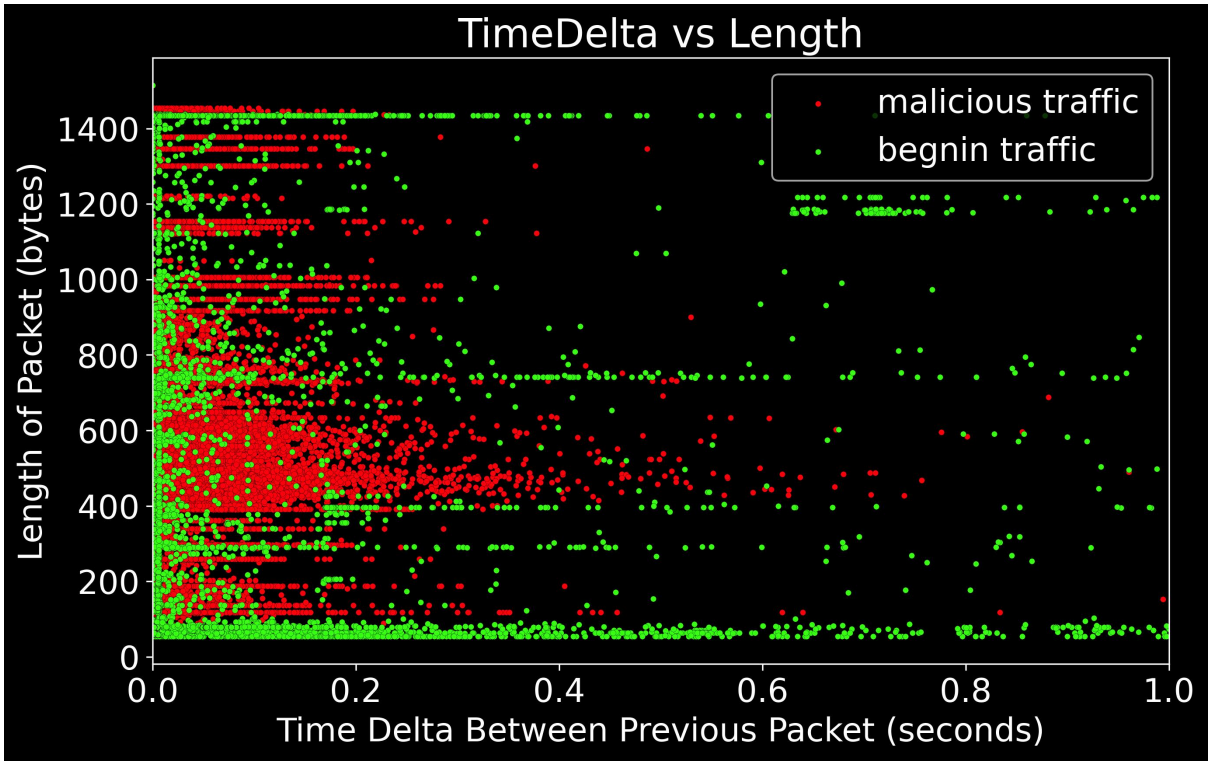

Figure 7: Time delta vs Length of packets

3.4 Rationale

To collect the network data, A DoS attack was simulated in a controlled environment on personal computers which would not affect any third party and also abides by all governmental laws and regulations. As shown in the scatter plot of the network traffic, the interval between consecutive malicious packets (shown in red) is typically in the 0.0–0.2 second range and of length 400–800. Therefore, it was hypothesised that with a fairly simple machine learning model, malicious traffic could be classified from the benign traffic. However, it is important to acknowledge that this may have arisen due to a bias of using the 'PyFlooder' tool, and another penetration testing tool could employ other attack patterns.

3.5 Normalization

In the field of machine learning, data normalization is a crucial process which transforms data into a form that can be understood and processed by machine learning models. This transformation of data to numerical representation aids the models in recognizing patterns and making accurate predictions. When it comes to IP addresses, each one is a unique identifier for a device on a network. However, these addresses are written in a format that is easy for humans to read, not machines. IP addresses are typically composed of four sets of numbers, each set ranging from 0 to 255, separated by periods (e.g., 192.168.1.1). For the machine learning model to make sense of these addresses, it was needed to convert or 'normalize' them into a single numerical value.

Therefore, normalization was achieved using a technique similar to what is used in computer programming called 'hashing'. Simply put, hashing takes in data of any size and spits out a fixed-size numerical 'hash'.

```python
def read_and_preprocess_csv():
    df = pd.read_csv("TrainData.csv")
    pd.set_option('display.float_format', lambda x: '%.32f' % x)
    # Calculate time delta between consecutive timestamps
    df['timestamp'] = df['timestamp'].diff()
    df['timestamp'] = df['timestamp'].fillna(0)

    # Normalize the data
    df['label'] = df['label'].map({'HTTP_flood_attack': 1, 'Legitimate_traffic': 0})

    protocol_mapping = {
        'DNS': 0,
        'HTTP': 1,
        'ICMP': 2,
        'TCP': 3,
        'UDP': 4
    }

    df['protocol'] = df['protocol'].replace(protocol_mapping)

    # Validate the IPs
    df = df[df['src_ip'].apply(lambda x: x.count('.') == 3 and all(0 <= int(i) <= 255 for i in x.split('.')))]
    df = df[df['dst_ip'].apply(lambda x: x.count('.') == 3 and all(0 <= int(i) <= 255 for i in x.split('.')))]
    df['dst_ip'] = df['dst_ip'].apply(lambda x: int(''.join([str(int(i)) for i in x.split('.')])) % (2**32))
    df['src_ip'] = df['src_ip'].apply(lambda x: int(''.join([str(int(i)) for i in x.split('.')])) % (2**32))

    # Normalize continuous columns to range [0, 1]
    continuous_cols = ['src_ip', 'dst_ip']

    df[continuous_cols] = df[continuous_cols] / (2**32 - 1)

    return df
```

Figure 8: Snippet of code used to normalize and validate the IPs

The hashing method used, as seen in the above figure, converted the entire IP address as a single number and applied a mathematical operation called modulo with the base of 2^32. This operation essentially limits our hash values to a fixed range, between 0 and 2^32 (ChatGPT). Then the data is normalized to a range from 0 to 1 as it is considered best practice when training a Neural Network.

One potential hiccup to be aware of with this method is called a 'hash collision', where two different IP addresses could theoretically produce the same hash value. However, given the large range provided by the chosen base (2^32), the chances of such a collision occurring are negligible. To elaborate, the number 2^32 was chosen as it is the total count of all possible IPv4 addresses. However, with IPv6 addresses, the total count jumps up to 2^128 and the chances of a hash collision would be much higher and the normalization process would be computationally unfeasible. Since no IPv6 addresses were present in the dataset, hash collision was not deemed as a significant problem.

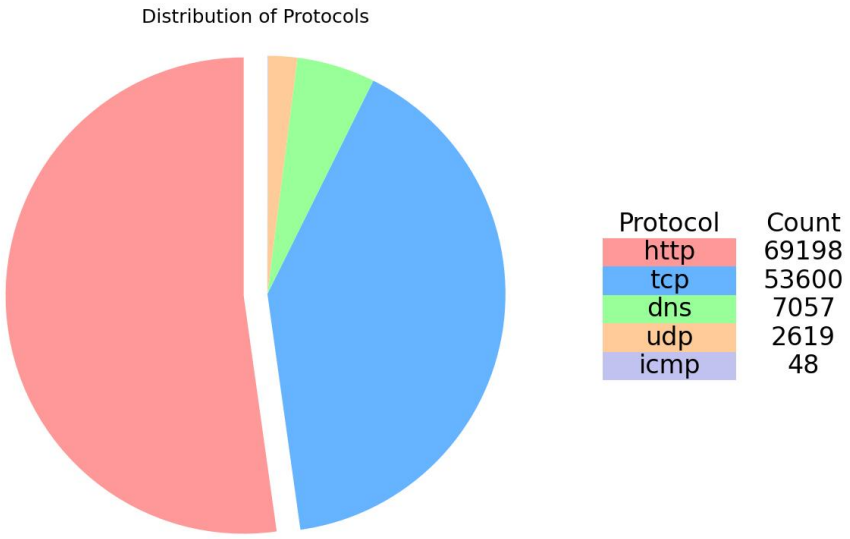

Figure 9: Distribution of Protocols in the dataset

Normalizing the protocols was a bit easier. Since there were only 5 protocols in the dataset, as shown in the above figure For the protocol field, a mapping approach was used to convert different protocol types into numerical identifiers. The following mapping was used: DNS is represented as 0, HTTP as 1, ICMP as 2, TCP as 3, and UDP as 4. The length attribute, which denotes the size of the packet, was retained in its original form as it is already a numerical value with an interpretable meaning. Lastly, the label field was binary-encoded to denote whether a packet is benign or malicious. A packet labelled as 'benign' was mapped to 0, while 'malicious' was mapped to 1. This approach facilitates a clear distinction between benign and malicious network traffic in our machine learning model. The same snipped of the data after the normalization processes is shown below, to 9 significant figures. While plenty to demonstrate the process, helps to prevent reverse engineering the original IP addresses of the author:

Normalized Network Data						
#	time delta	source IP	destination IP	protocol	length	label
0	00.000000	4.00844263E-4	2.34909100E-1	1	448	1
1	00.756626	4.00844300E-4	2.34909100E-1	1	468	1
2	02.259044	4.00844263E-4	2.34909100E-1	1	480	1
3	07.316074	4.00844263E-4	2.34909100E-1	1	572	1
4	00.606636	4.00844263E-4	2.34909100E-1	1	632	1
...	...	...	...	...	...	...
132546	18.957310	4.47426340E-2	4.47426247E-3	0	64	0
132547	00.000200	4.47426247E-3	4.47426340E-2	0	59	0
132548	00.001040	4.47426340E-2	9.83243808E-7	0	64	0
132549	00.005130	9.83243808E-7	4.47426340E-2	0	59	0
132550	00.001120	4.47426340E-2	9.83476639E-7	0	64	0

Table 2: Normalized Data

3.6 Neural Networks

This research utilizes TensorFlow, a widely used open-source python library for machine learning, to create a sequential Neural Network. This Neural Network is designed to classify HTTP traffic as either benign or malicious, based on the features extracted from the dataset, namely: time delta, source IP, destination IP, protocol, and packet length.

The model used is called a sequential model, because the layers of neurons in the network are organized linearly, with each layer passing its output forward to the next layer (fchollet). The model consists of 4 layers, as shown in Figure 1. The second and third layers are 'dense' layers, each with 32 and 16 neurons, respectively. In a dense layer, each neuron is connected to every neuron in the previous layer, allowing for complex patterns to be learned from the data.

```
25  # Build the neural network model
26  model = tf.keras.models.Sequential([
27      tf.keras.layers.Dense(32, activation='relu', input_shape=[5]),
28      tf.keras.layers.Dense(16, activation='relu'),
29      tf.keras.layers.Dense(1, activation='sigmoid')
30  ])
31
32  # Compile the model
33  model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
34
35  # Prepare callbacks for model saving and for learning rate adjustment.
36  checkpoint_cb = callbacks.ModelCheckpoint(
37      'my_model.h5', save_best_only=True)
38
39  early_stopping_cb = callbacks.EarlyStopping(
40      patience=10,
41      restore_best_weights=True)
42
43  # Train the model
44  NeuralNetwork = model.fit(
45      train_dataset.batch(16),
46      epochs=20,
47      validation_data=test_dataset.batch(16),
48      callbacks=[checkpoint_cb, early_stopping_cb])
49
```

Figure 10: Code used to create the model

The activation function used in these dense layers is the Rectified Linear Unit (ReLU) (Keras). The purpose of an activation function is to introduce nonlinearity into the model, allowing it to learn and represent more complex patterns. ReLU is a popular choice due to its simplicity and efficiency. It simply sets all negative input values to zero and leaves positive values unchanged. This function is known to help mitigate the 'vanishing gradients' problem, a common issue in training Neural Networks. To put it simply, the vanishing gradient problem arises when a Neural Network finds a local minimum in loss function. It can be compared to a child who is unwilling to step out of their comfort-zone and pursue new endeavours, which limits the child's potential. Similarly, the Neural Network stops making improvements once it reaches a local minimum.

The fourth and final layer in our model is also a dense layer, but it consists of a single neuron and uses a different activation function, the sigmoid function. The sigmoid

function squashes its input into a range between 0 and 1, effectively allowing it to output probabilities (Keras). Since our task is a binary classification (malicious or benign), this is a fitting choice. The output of the sigmoid function can be interpreted as the probability of the traffic being malicious or benign.

When training the Neural Network, The model undergoes multiple iterations of exposing the entire dataset to the model. Each iteration is called an 'epoch'. During each epoch, the model makes predictions based on the input data, calculates how much these predictions deviate from the actual labels (the 'loss'), and then adjusts the weights of the neurons in the network to minimize this loss.

In order to prevent overfitting, which occurs when a model learns the training data too well and performs poorly on unseen data, A technique called 'early stopping' is employed. This stops the training process if the model's performance on a validation set (a portion of the training data set aside for this purpose) stops improving.

4 Results

The graph below, shows the accuracy and validation accuracy over each epoch. At the end of the training process, an accuracy of 95.4% was noted. This showcased a success as the machine learning model had achieved a high degree of accuracy.

While the accuracy of the Neural Network had begun to flatline by its tenth epoch, validation accuracy, which is a more important metric which cross–checks the Neural Network with a segment of the data it has never seen before. Therefore, dips in this metric was expected but the eventual flatlining, and complete overlap with accuracy, by its thirteenth epoch, suggests that the Neural Network was sufficiently trained, and it is doubtful that it would improve beyond this point.

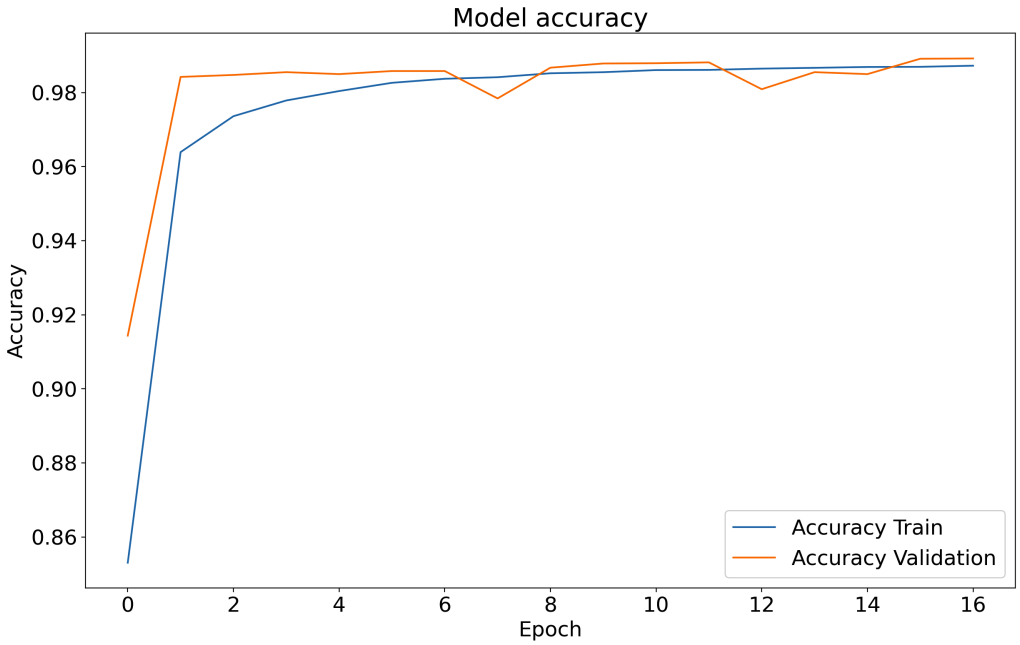

Figure 11: Accuracy of the Neural Network, over 20 epochs

Now that it has been established that the Neural Network was done learning, this is the confusion matrix for the predictions it made.

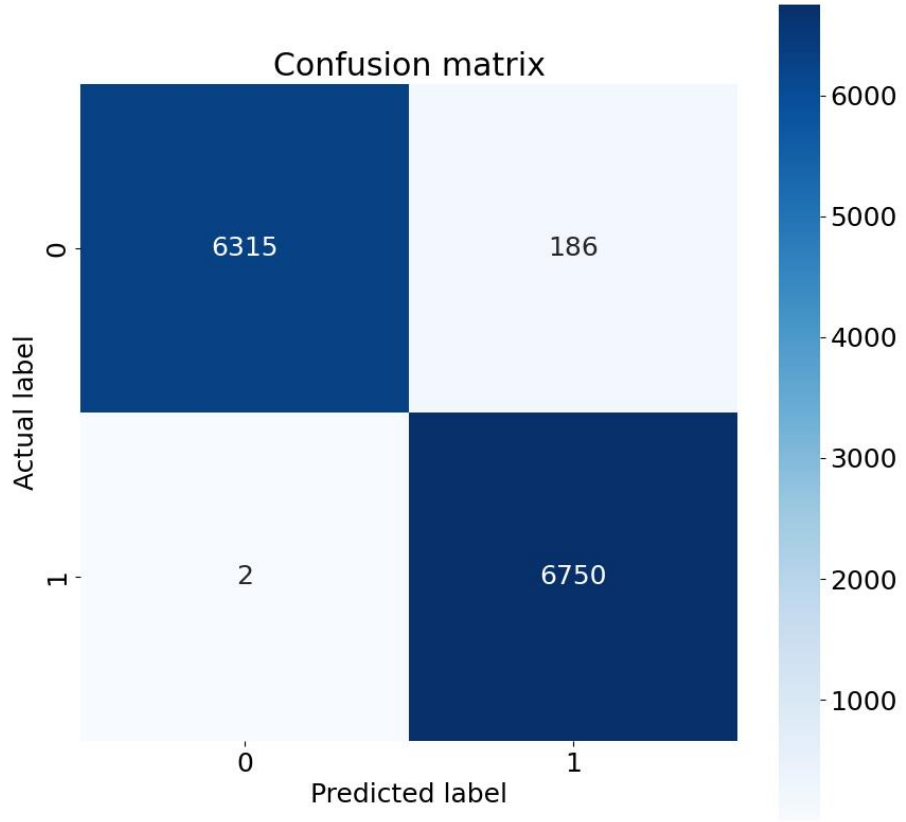

Figure 12: Confusion matrix for the Neural Network

The above confusion matrix showcases that, while, the model does make the occasional false positive, the relatively low number of false negatives is far more important as it is better to flag benign traffic than let malicious traffic flow, undetected.

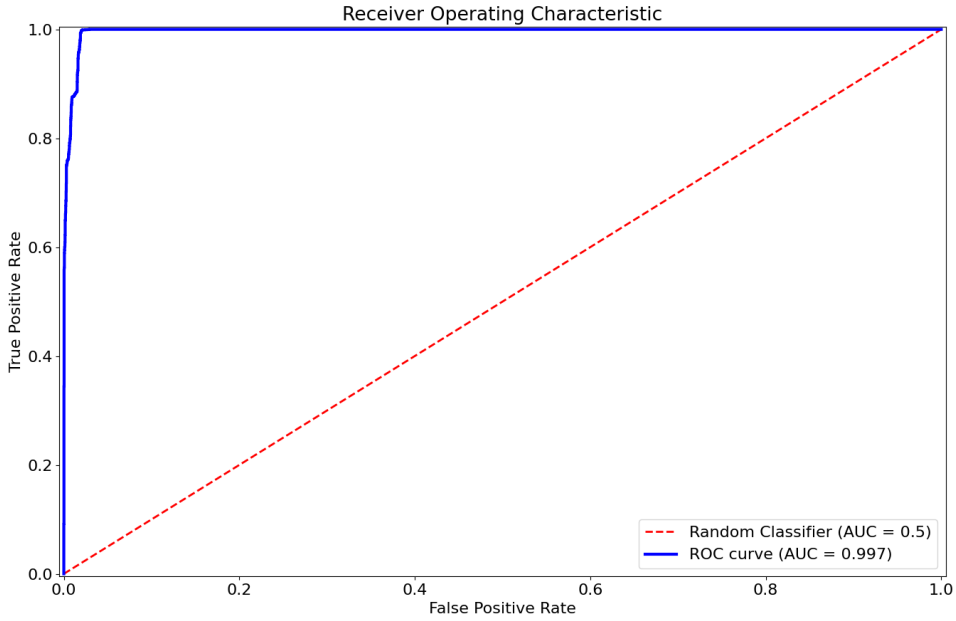

Figure 13: Receiver Operature Characteristics for the Neural Network

The model's capacity to maintain a low false positive rate while achieving almost perfect true positive rate, showcases its effectiveness. As the ROC curve approaches the upper-left corner, it signifies an optimal balance between true positive and false positive rates, reflecting a model with exceptional discriminative capabilities. As the model had a FPR of 2.68% and a TPR of 99.97%, Hence, the ROC curve approaches the left corner, but a gap remains, with the y-axis due to a true negative rate of 97.32 (1-FPR), which is smaller, compared to the TPR of 99.97%.

5 Limitations

While the results paint a glowing picture of the machine learning model, it was not fine-tuned, and it is estimated that performance could further improve with fine-tuning of the hyperparameters of the Neural Network. It is also possible that the simple Neural Network used, performed well due to the limited number of parameters in the dataset. However, in a real world scenario, it is possible that the limited features of the dataset, could limit the Neural Network's performance, and consequently the Neural Network could be too simple to analyze the more complex dataset. Furthermore, 'as is the case with most classifiers based on Neural Networks, may not calibrate well outside its training data' (OpenAI). Specifically, given that the primary training was centred around HTTP GET flood, the efficacy of the model might decrease when confronted with detecting malicious HTTP POST or even slower HTTP GET DoS attacks. This hypothesis was tested using an open-source dataset 'with a focus on HTTP DoS attacks' (Reed). The Neural Network model was tested against this unknown dataset (Reed) without any further training and there was a drop in accuracy from 95.4% to 92.2%.

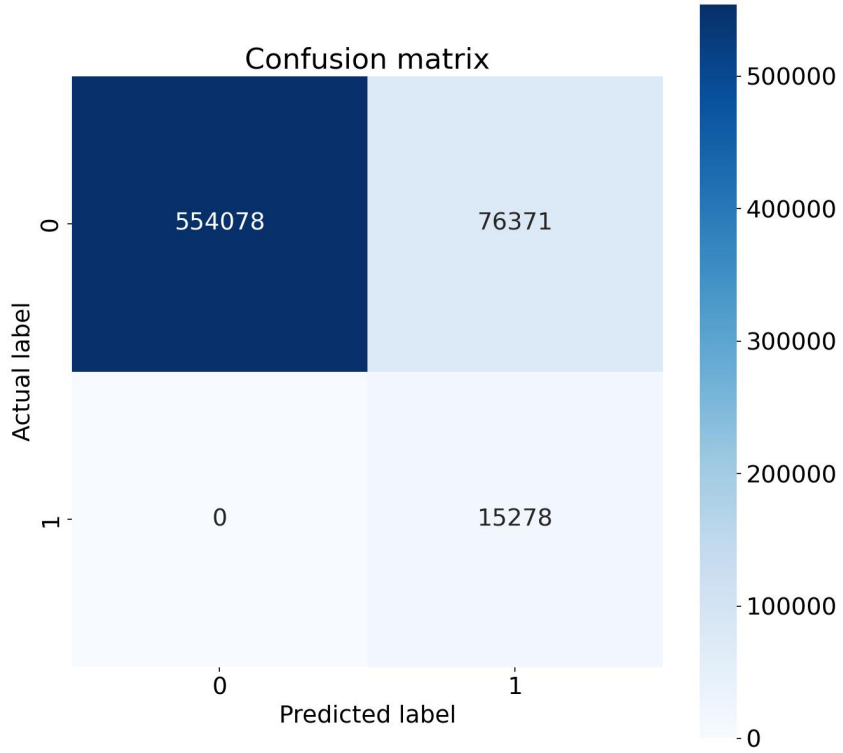

Figure 14: Confusion matrix of the machine learning model for the unknown dataset

As shown above, the Neural Network still performs to an exceptional degree of accuracy, but this experiment helps us understand the extent to which the machine learning model is able to perform. Another major limitation is the data processing pipeline, which would benefit the most from automation. For example, Deep Learning could be employed for feature extractions which would normally be done by a human being. The benefit to this approach would be faster, robust and more qualitative data, therefore giving the model more information to work with, which may lead to higher accuracy. However, that is out of scope for this research.

6 Future Scope

As stated above, one key area of improvement to the approach taken in this Extended Essay would be to utilize Deep Learning for feature extraction. Therefore, automatically processing the raw data, completely mitigating the need for human intervention. As a result, potential errors due to human biases, can be reduced. This was omitted in this research for the sake of simplicity.

Secondly, a model that is exposed to a bigger variety of DoS malware signatures, would ensure feasability of using such a DoS attack detection technique in the real world.

The implementation of these improvements would also open doors to real time processing and mitigation of threats, which could possibly enable deployment of the model in real world situations.

7 Conclusion

This study delved into the application of machine learning, specifically Neural Network-based binary classifiers, in detecting HTTP flood attacks. The research journey encompassed data collection, feature extraction, model training, and analysis.

The dataset, with a total count of 132,550 packets, was relatively balanced, comprising 66,937 malicious packets and 65,613 benign ones. A sequential model architecture was employed for the classifier with five input nodes corresponding to the features and a single output node indicating the classification. The model utilized the 'Adam' optimization algorithm for the backpropagation process and the 'Sigmoid' function to calculate the probability of a packet being malicious, therefore fullfilling its role, as a classifier.

The results demonstrated the Neural Network's proficiency in distinguishing between benign and malicious traffic, achieving an accuracy of 95.4%. However, when exposed to a different dataset, the accuracy slightly declined to 92.2%.

The study highlighted the need for ongoing adaptation and fine-tuning, considering the sensitivity to specific attack types and the potential for false positives. The significance of automating data processing was also underscored for enhancing accuracy and efficiency.

In summary, machine learning and Neural Networks offer a promising path for strengthening cybersecurity. While achieving notable accuracy, the research emphasized the importance of continuous refinement to address evolving threats, aiming for a more secure digital landscape.

Works Cited

Abuse of vulnerabilities. *Akamai*, Aug. 2023. https://www.akamai.com/newsroom/press-release/akamai-research-rampant-abuse-of-zero-day-and-one-day-vulnerabilities-leads-to-143-increase-in-victims-of-ransomware. Accessed 20 Aug. 2023.

Alexlenail. Publication-ready NN-architecture schematics. *Alexlenail.me*, 2023. https://alexlenail.me/NN-SVG/. Accessed 23 Aug. 2023.

ChatGPT. "Prompt: "What is hashing?" August 21 Version". *OpenAI*, 2023. Accessed 23 Aug. 2023.

Cloudflare. HTTP flood attack. https://www.cloudflare.com/learning/ddos/http-flood-ddos-attack/. Accessed 18 Aug. 2023.

Confusion Matrix. *scikit-learn*, 2023. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html. Accessed 18 Aug. 2023.

Delplace, Antoine, et al. Cyber Attack Detection thanks to Machine Learning Algorithms. 2020. *arXiv*, https://arxiv.org/pdf/2001.06309.pdf. Accessed 18 Aug. 2023.

fchollet. The Sequential model. Apr. 2020. https://keras.io/guides/sequential_model/. Accessed 18 Aug. 2023.

Gad, Ahmed Fawzy. Deep Learning Metrics: Precision, Recall, Accuracy. 2020. https://blog.paperspace.com/deep-learning-metrics-precision-recall-accuracy/. Accessed 18 Aug. 2023.

Google. Classification: ROC Curve and AUC. 2022. https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc. Accessed 18 Aug. 2023.

Keras. Layer activation functions. 2020. https://keras.io/api/layers/activations/. Accessed 18 Aug. 2023.

Kirichenko, David. Crowdsourced Cyber Warfare: Russia and Ukraine Launch Fresh DDoS Offensives. *CEPA*, July 2023. https://cepa.org/article/russia-ukraine-launch-cyber-offensives/. Accessed 18 Aug. 2023.

Medina, Gonzalo. How to construct a confusion matrix in LaTeX? *TeX - LaTeX Stack Exchange*, June 2011. https://tex.stackexchange.com/questions/20267/how-to-construct-a-confusion-matrix-in-latex. Accessed 18 Aug. 2023.

OpenAI. *AI Text Classifier*, Jan. 2023. https://beta.openai.com/ai-text-classifier. Accessed 18 Aug. 2023.

Primusa and Christian Vincenzo Traina. What is the difference between the terms accuracy and validation accuracy. May 2021. https://stackoverflow.com/questions/51344839/

what-is-the-difference-between-the-terms-accuracy-and-validation-accuracy. Accessed 18 Aug. 2023.

PyFlooder. *GitHub*, June 2021. https://github.com/D4Vinci/PyFlooder. Accessed 18 Aug. 2023.

Reed, Andy. "HTTP DoS Dataset in PCAP format for Wireshark". *figshare*, Dec. 2021. https://ordo.open.ac.uk/articles/dataset/HTTP_DoS_Dataset_in_PCAP_format_for_Wireshark/17206289/1. Accessed 18 Aug. 2023.

Smilkov, Daniel, and Shan Carter. Neural Networks. 2023. https://playground.tensorflow.org/. Accessed 18 Aug. 2023.

Wireshark. What is Wireshark? 2023. https://www.wireshark.org/faq.html#_what_is_wireshark. Accessed 18 Aug. 2023.

Works, Math. Machine learning. 2023. https://de.mathworks.com/discovery/machine-learning.html. Accessed 18 Aug. 2023.

A Appendix

```
import pandas as pd
import re
import matplotlib.pyplot as plt
from matplotlib.pyplot import subplot # typehinting

def read_and_preprocess_csv():
    df = pd.read_csv("TrainData.csv")
    pd.set_option('display.float_format', lambda x: '%.32f' % x)
    # Calculate time delta between consecutive timestamps
    df['timestamp'] = df['timestamp'].diff()
    df['timestamp'] = df['timestamp'].fillna(0)

    # Normalize the data
    df['label'] = df['label'].map({'HTTP_flood_attack': 1, '
    Legitimate_traffic': 0})

    protocol_mapping = {
        'DNS': 0,
        'HTTP': 1,
        'ICMP': 2,
        'TCP': 3,
        'UDP': 4
    }

    df['protocol'] = df['protocol'].replace(protocol_mapping)

    # Validate the IPs
    df = df[df['src_ip'].apply(lambda x: x.count('.') == 3 and all(0 <=
    int(i) <= 255 for i in x.split('.')))]
    df = df[df['dst_ip'].apply(lambda x: x.count('.') == 3 and all(0 <=
    int(i) <= 255 for i in x.split('.')))]
    df['dst_ip'] = df['dst_ip'].apply(lambda x: int(''.join([str(int(i))
    ) for i in x.split('.')])) % (2**32))
    df['src_ip'] = df['src_ip'].apply(lambda x: int(''.join([str(int(i))
    ) for i in x.split('.')])) % (2**32))

    # Normalize continuous columns to range [0, 1]
    continuous_cols = ['src_ip', 'dst_ip']

    df[continuous_cols] = df[continuous_cols] / (2**32 - 1)

    return df

def plot_protocol_distribution(df):
    # Mapping of protocols
    protocol_mapping = {
        0: 'dns',
        1: 'http',
        2: 'icmp',
        3: 'tcp',
        4: 'udp'
    }

    # Count the occurrences of each protocol
    protocol_counts = df['protocol'].map(protocol_mapping).value_counts
    ()
```

```
51
52      # Plot the pie chart
53      fig, ax = plt.subplots(figsize=(10, 6), ncols=2, gridspec_kw={'
        width_ratios': [2, 1]})
54      colors = ['#ff9999', '#66b3ff', '#99ff99', '#ffcc99', '#c2c2f0'][:
        len(protocol_counts)]
55      explode = [0.1 if protocol == 'http' else 0 for protocol in
        protocol_counts.index]
56      wedges, *_ = ax[0].pie(protocol_counts, labels=None, colors=colors,
57                             startangle=90, explode=explode)
58      ax[0].axis('equal')
59
60      # Set white background color for the pie chart
61      ax[0].set_facecolor('white')
62
63      # Create the table
64      table_data = list(zip(protocol_counts.index, protocol_counts))
65      table = ax[1].table(cellText=table_data, colLabels=['Protocol', '
        Count'], colWidths=[0.5, 0.5],
66                          cellLoc='center', loc='center')
67      table.set_fontsize(16)
68      table.scale(1, 1.5)
69
70      # Remove table borders
71      for key, cell in table.get_celld().items():
72          cell.set_linewidth(0)
73
74      # Set text color in the table to black
75      for cell in table.get_celld().values():
76          cell.get_text().set_color('black')
77
78      # Highlight the protocol name and set its color
79      for row in range(1, len(table_data) + 1):
80          protocol_name = table[(row, 0)].get_text().get_text()
81          cell_color = colors[row-1]
82          table[(row, 0)].set_facecolor(cell_color)
83
84      # Remove unnecessary axes in the table subplot
85      ax[1].axis('off')
86
87      # Add a title to the pie chart
88      ax[0].set_title('Distribution of Protocols')
89      plt.savefig('ProtocolPlot.jpg', bbox_inches='tight', dpi=150)
90
91  def scatter_plot(dataframe: pd.DataFrame):
92
93          http_flood_data = dataframe[dataframe['label'] == 1]
94          legitimate_data = dataframe[dataframe['label'] == 0]
95
96          plt.style.use('dark_background')
97          plt.rcParams.update({'font.size': 18}) #Make it easier to read
98          plt.figure(figsize=(10, 6))
99          plt.scatter(http_flood_data['timestamp'], http_flood_data['
        length'], color='#FF000D', edgecolor='black', linewidth=0.1, s=10,
        label='malicious traffic')
100         plt.scatter(legitimate_data['timestamp'], legitimate_data['
        length'], color='#39FF14', edgecolor='black', linewidth=0.1, s=10,
        label='benign traffic')
```

```
        plt.xlim(-0.0001, 1)
        plt.xlabel('Time Delta Between Previous Packet (seconds)')
        plt.ylabel('Length of Packet (bytes)')
        plt.title('TimeDelta vs Length')
        plt.legend()
        plt.savefig('TrafficScatterPlot.jpg', bbox_inches='tight', dpi
    =300)

preprocessed_df = read_and_preprocess_csv()
plot_protocol_distribution(preprocessed_df)
scatter_plot(preprocessed_df)
preprocessed_df.to_csv('TrainDataTreated.csv')
```

Algorithm 1: Normalization and Diagrams

```
import pandas as pd
import tensorflow as tf
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from keras import callbacks
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import confusion_matrix

# Load dataset into a pandas dataframe
df = pd.read_csv(r'TrainDataTreated.csv')

# Split the dataset into features and labels
features = df[['timestamp', 'src_ip', 'dst_ip', 'protocol', 'length']]
labels = df['label']

# Split data into training and test sets
train_features, test_features, train_labels, test_labels =
    train_test_split(features, labels, test_size=0.1, random_state=42)

# Convert to TensorFlow Datasets
train_dataset = tf.data.Dataset.from_tensor_slices((train_features.
    values, train_labels.values))
test_dataset = tf.data.Dataset.from_tensor_slices((test_features.values
    , test_labels.values))

# Build the neural network model
model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(32, activation='relu', input_shape=[5]),
    tf.keras.layers.Dense(16, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['
    accuracy'])

# Prepare callbacks for model saving and for learning rate adjustment.
checkpoint_cb = callbacks.ModelCheckpoint(
    'my_model.h5', save_best_only=True)

early_stopping_cb = callbacks.EarlyStopping(
    patience=10,
    restore_best_weights=True)

# Train the model
NeuralNetwork = model.fit(
    train_dataset.batch(16),
    epochs=20,
    validation_data=test_dataset.batch(16),
    callbacks=[checkpoint_cb, early_stopping_cb])

# Save the model for later use
model.save('my_model.keras')

# Make Predictions after training is complete
predictions = model.predict(test_features)
```

```
# Round off the predictions to get binary classification
rounded_predictions = [round(x[0]) for x in predictions]

plt.rcParams.update({'font.size': 18}) #Make it easier to read

# Plot the confusion matrix
cm = confusion_matrix(test_labels, rounded_predictions)
plt.figure(figsize=(10, 10))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', square=True)
plt.title('Confusion matrix')
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
plt.savefig('ConfusionMatrix.jpg', format='jpg', bbox_inches='tight')
plt.show()

# Plot the ROC curve + AUC value
fpr, tpr, threshold = roc_curve(test_labels, predictions)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(10, 6))


# Plot the random classifier
plt.plot([0, 1], [0, 1],
         color='red',
         lw=2,
         linestyle='--',
         label='Random Classifier (AUC = 0.5)')

# Plot the ROC curve for the trained classifier
plt.plot(fpr, tpr,
         color='blue',
         lw=3,
         label='ROC curve (AUC = %0.3f)' % roc_auc)

plt.xlim([-0.005, 1.005]) # Set axes limits
plt.ylim([-0.005, 1.005])

# Add axes
ax = plt.gca()
ax.spines['bottom'].set_color('black')
ax.spines['left'].set_color('black')
ax.xaxis.label.set_color('black')
ax.yaxis.label.set_color('black')
ax.tick_params(axis='x', colors='black')
ax.tick_params(axis='y', colors='black')

plt.tight_layout()
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc='lower right')
plt.savefig('ROC.jpg', bbox_inches='tight', dpi=150)
plt.show()

# Plot Accuracy
plt.figure(figsize=(10, 6))
plt.tight_layout()
```

```
plt.xticks(np.arange(0,21,2),np.arange(0,21,2))
plt.plot(NeuralNetwork.history['accuracy'])
plt.plot(NeuralNetwork.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Accuracy Train', 'Accuracy Validation'], loc='upper left')
plt.savefig('Accuracy.jpg', bbox_inches='tight', dpi=150)
plt.show()

# Plot Loss
plt.figure(figsize=(10, 6))
plt.tight_layout()
plt.xticks(np.arange(0,21,2),np.arange(0,21,2))
plt.plot(NeuralNetwork.history['loss'])
plt.plot(NeuralNetwork.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Loss Train', 'Loss Validation'], loc='upper left')
plt.savefig('Loss.jpg', bbox_inches='tight', dpi=150)
plt.show()

# Save the metrics throughout the epoches for evaluation
with open('history.txt', 'w+') as file: #Overwrite data from previous
    tests
    file.write(str(NeuralNetwork.history)+'\n')
    file.write((f'False positive rate: {list(fpr)}, True Positive Rate:
    {list(tpr)}, Threshold: {list(threshold)}, AUC: {roc_auc}'))
    file.close()
```

Algorithm 2: Machine Learning Model and Metrics