Detecting Distributed Denial-of-Service Attacks on Networks

Jack Davies
School of Computer Science & Informatics, Cardiff University, UK
MSc Cyber Security & Technology
Master's Thesis
Supervisor: Dr. Amir Javed
Moderator: Dr. George Theodorakopoulos

1 INTRODUCTION

Over the last two decades, we have seen a massive shift towards more online connectivity. With 25% of all business in the USA being conducted online (IBISWorld, 2022), and similar patterns seen in much of the world, the Internet has gone from a useful resource to a near necessity; with many small, medium, and even large businesses relying on business from their Web-based customers. In the early 2000s, less than 7% of the world was online - today, that number is over 60% (Roser et al., 2020) and this is only expected to increase. This level of online activity has benefited many people. Amazon now employs more than 1.3 million people worldwide (Amazon, 2022). There are over 50 million business pages (Chaykowski, 2015) and nearly 3 billion users on Facebook (Dixon, 2022), allowing unprecedented customer reach and generating many new small businesses due to the small startup costs. In fact, the E-commerce revenue in the UK has risen from £42.6 billion to £129 billion in just 6 years (Coppola, 2022).

However, these benefits come with drawbacks. We now rely on Internet connectivity, the Covid-19 pandemic highlighted the need to be able to communicate with each other, in a personal as well as business respect. The connectivity provided by the Internet is crucial for almost everything we do today. This reliance puts us in a tenuous situation where the loss of this connectivity could have dramatic consequences.

Distributed Denial of Service (DDoS) attacks are a threat to this connectivity. A DDoS attack overwhelms a server's ability to respond to requests, thus preventing access to legitimate users. In Q1 of 2022, there was a reported 1,406 DDoS attacks per day (Gutnikov et al., 2022), this is a 450% rise compared to the same period last year - largely attributed to the Russia-Ukraine conflict. The shared *cyberspace* has become a primary battleground in modern conflict, thus it is important to be always progressive in our ability to respond to these attacks.

In order to effectively respond to DDoS attacks, one must be able to recognise when an attack is occurring. This report investigates a novel method of identifying DDoS attacks using a hybrid machine learning system. We train an unsupervised anomaly-detection model to recognise when an attack may be taking place. The packet flows associated with a potential attack are then passed to a supervised classification model, which classifies the attack into a known class. If classification is not possible, the data is separated for further investigation and fed back into the system to iteratively improve it.

In this report, the current state of DDoS attack recognition is investigated and reviewed and the overall problem is outlined. This is followed by a high-level description of the proposed solution. A review of potential machine learning techniques that could be used is then provided, with a rationale for the methods ultimately chosen. The prototype implementation of the proposed system is described and experimental methodology is then explained and the results of these experiments are provided with conclusions drawn from said experiments. Following this, potential future work is outlined and finally, a reflection on the project is given.

2 RELATED WORK

Brihat Ratna Bajracharya (Bajracharya, 2019) presents a method of detecting DDoS attacks on a set of packet data using logistic regression. Logistic regression uses one or more independent variables to determine the probability that an object belongs to

a particular class. The author trains and tests their models using the CICDDoS2019 data set (Canadian Institute for Cybersecurity, New Brunswick, 2019), specifically the PortMap and LDAP sets, which show packet flow data of abstract behaviour and whether a packet flow was linked to a DDoS attack or not.

In this paper, the author uses a binomial logistic regression for the PortMap model, where classification is determined by setting a threshold value for the probability (in this case, 0.5) that defines if the outcome is class 1 where the threshold is exceeded, or class 0 otherwise - where 1 is the positive class, being a PortMap attack, and 0 is benign. The author uses a multinomial logistic regression for the LDAP classification model. where the probability of each of three potential classes (LDAP, NetBIOS and benign) is given by an output vector with a set of probabilities summing to 1.

The logic for using such methods can be clearly seen, the problem of DDoS attack recognition can be described as a binary classification problem - regardless of the type or protocol, *is this set of packets part of a DDoS attack (1), or not (0)?* Ultimately, the number of classes that can be recognised by a given model is irrelevant when trying to determine this binary outcome. The method shown in this paper can therefore be used to detect a range of different types of DDoS attacks. The results of the experiment in the paper show that the trained classification models appear to have a strong accuracy in determining the presence of a DDoS attack from a flow of packets, with 99.91% accuracy for PortMap attack detection and 99.94% for LDAP.

The paper is perhaps lacking in application in that it is only used in classification on a static set of data and has never been tested on a live system, it would be interesting to see how this sort of model performs when dealing with a live system. The source port and IP address are recognised in this paper as being two of the strongest packet-flow features in the training of the PortMap classification model and this could suggest that an attack from a new source would not be recognised as part of a DDoS attack.

Sahi et al. (Sahi et al., 2017) describe a DDoS detection and prevention system from TCP flood attacks on cloud environments. The system is split into two subsystems: detection and prevention. The detection part of the system is based on a classification algorithm, where an incoming packet on a live network is classified as being either part of a DDoS attack or not. Packets determined to be part of a DDoS attack are sent to a prevention subsystem which blacklists the source IP address and drops the incoming packets from any blacklisted IP.

The detection subsystem was tested using multiple classification algorithms to determine the most reliable: LS-SVM, Naïve Bayes, *k*-nearest and Multilayer Perceptron. The data used to train and test the classifiers was self-created and used a bespoke algorithm to classify whether a packet was part of a DDoS attack or not - this being if the number of packets from the same source IP address within 60 seconds exceeded a user-defined threshold. The classifiers were given two packet features as input variables, source IP and destination IP. Ultimately, LS-SVM was determined to be the most reliable classifier in this experiment, with 97% accuracy, 97% sensitivity and 97% specificity.

The results of the experiment appear to suggest strong reliability of the detection subsystem. However, it should be noted that whilst the use of a user-defined threshold for the number of packets within 60 seconds as a method of recognising DDoS attacks could be useful for certain situations, it is ultimately an arbitrary number, and could easily be calibrated incorrectly. Furthermore, it does highlight that only one type of DDoS attack can be prevented with this system – although that is the scope of the paper from the beginning. The use of only the source IP and destination IP as input features for the classifier also suggests that it cannot be used to detect DDoS attacks where a spoofed source IP address is used.

That said, the paper does show a viable method for dealing with DDoS attacks, there is no obvious reason why the prevention subsystem could not be used for responding to different types of DDoS attacks and it could be expanded on further to be more nuanced in its response. In addition, it is positive that the paper shows the system working on a live network and it can deal with individual packets rather than a flow, potentially preventing DDoS attacks before they have even begun. The comparison of different classifiers is also useful in this research and suggests possible alternatives to the aforementioned logistic regression.

Balkanli et al. (Balkanli et al., 2014) make comparisons between existing network intrusion detection systems (NIDS) and two different classifiers in their ability to detect DDoS attacks. Along with two well-known NIDS, the authors create a CART decision tree classifier and a Naïve Bayes classifier and measure the reliability of DDoS detection on a set of backscatter data from darknet channels – the rationale for this is that darknet addresses should not receive any legitimate traffic as there is no service running on these networks, thus all traffic directed at these addresses can be considered part of a malicious attack.

The NIDS used in this paper use a signature-based detection method for DDoS attack detection, whereas

the classifiers naturally use a machine learning approach. The authors identify two sets of features from the data for the classifiers to learn from, the first set contains IP addresses and source/destination port numbers, and the second set omits these.

The results show that the two classifiers performed as well as or better than the two NIDS, with the decision tree classifier slightly outperforming the Naïve Bayes classifier in reliability and vastly outperforming it in speed. Interestingly, the results showed that using a smaller training set was beneficial in that it reduced training time and improved accuracy. Furthermore, the second feature set which omitted IP addresses and port numbers had comparable or better reliability – suggesting that this information is not required to detect DDoS attacks, an important point when wanting to generalise a detection system to recognise attacks from new sources. The authors highlight that the protocol, ACK flag and RST flag are features from packets that appear to be the most important in identifying DDoS attacks.

The runtimes of the classifiers used in this paper are significant and may not be suitable for a live system. However, it should be noted that the majority of this time is spent in the training phase; it should only be required once per model.

Bindra and Sood (Bindra and Sood, 2019) provide an analysis of different machine learning classification algorithms for detecting DDoS attacks based on recent data. The notable importance of using recent, real-world data is that DDoS attacks have changed and become more sophisticated, thus older data sets that are commonly used for such tasks are now outdated. The authors compare five supervised classifiers: Support Vector Machine (SVM), Naïve Bayes, *k*-Nearest Neighbour (*k*-NN), Random Forest and Logistic Regression and train and test the models using the CICIDS2017 data set (Canadian Institute for Cybersecurity, New Brunswick, 2017b).

The results show that the most accurate classifiers used the *k*-NN and Random Forest algorithms, with the Random Forest classifier being slightly better than *k*-NN in the reliability of DDoS detection.

Unfortunately, the authors do not provide specific details on the data set and which features were used in the experiments, although it is stated that the number of features was reduced from 85 to 12. They also acknowledge that only one technique for data pre-processing was used, and this could have created a bias in their results, as certain machine learning models can be boosted by specific data pre-processing techniques. Nonetheless, it is interesting that the Random Forest was identified as the optimal classifier for detecting DDoS attacks. The Random Forest classifier is essentially a collection of polled decision trees running concurrently, and decision trees have been previously highlighted as reliable classifiers for DDoS detection in other related works.

Yan Li and Yifei Lu (Li and Lu, 2019) use a method of combining two classifiers to improve the classification of benign and DDoS attack packet data. The authors use a Long Short-Term Memory (LSTM) classifier with a Naïve Bayes (NB) classifier, in a combination dubbed LSTM-BA.

The LSTM classifier represents predicted results in a similar way to a logistic regression, where the Sigmoid function is used to make a prediction – a predicted value less than 0.5 is classified as normal traffic, and a value above this number is classified as attack traffic. Classification becomes unreliable for values close to 0.5. Therefore, the authors integrate the NB classifier to provide a "second opinion" on the more ambiguous results from the LSTM classifier. Their work shows that values from the LSTM classifier between 0.2 and 0.8 have a classification accuracy of just 74%, which is much lower than required. However, when these instances are passed to the NB classifier, the accuracy, precision and sensitivity scores hit around 98% in ultimate classification accuracy. Thus, a multi-layered approach using different types of classification algorithms appears to be useful, providing a much more reliable output.

Shieh et al. (Shieh et al., 2021) propose a hybrid-ML framework for identifying known and novel DDoS attacks which utilises a supervised Bi-Directional Long Short-Term Memory (BI-LSTM) model and an unsupervised Gaussian Mixture Model (GMM) with applied iterative learning. The authors recognise the issue with having a model that *"has no built-in mechanism for knowing what it does not know"* – the Open Set Recognition problem. This is an issue for DDoS attack recognition as new attacks can be created and performed at any time, so it is crucial that a detection system can quickly learn to respond to these.

The BI-LSTM model discriminates between benign traffic and DDoS attack traffic, whilst the GMM discriminates between learned samples and novel traffic. Novel traffic is forwarded to a 'Traffic Engineer' who then analyses and labels this traffic as either a new type of attack or as benign, this data is then fed back into the system to retrain the model – a process known as incremental or iterative learning. Thus, known attacks and unknown attacks have the potential to be recognised by the system, and recognition capabilities are flexible and continue to be improved with use.

The authors use CIC-IDS2017 and CICD-

DoS2019 data sets as the initial training sets for their model. The BI-LSTM model is tested first by comparing the classification accuracy on benign and attack traffic from two different sets – the first with "known" data (that is, data the model was trained on) and the second with novel data. Results show a poor performance in the reliability of classification on the second data set, with around a 50% drop in performance, sensitivity and accuracy. The authors then integrate an unsupervised GMM module where unknown traffics that breach a given threshold of deviation away from the mean, are considered novel instances. The traffic is labelled by a human operator and fed back into the system. After the completion of the loop, the results for the same experiment show much-improved classification reliability, with an improvement in sensitivity, precision, and accuracy, each to around 99.8%.

3 DETECTING DDOS ATTACKS

It is widely considered that prevention is better than cure. This applies to cyber security also; preventing an attack is far preferable to responding to one. There are many techniques to prevent DDoS attacks. One commonly-used technique is to apply ingress and egress filtering to the gateway of a network. Ingress filtering (Ferguson and Senie, 1998) involves filtering malicious incoming traffic by preventing access to any IP address not in a predefined range, egress filtering (Killalea, 2000) works in essentially the same way for outbound traffic. Such methods help prevent packets with spoofed IP addresses (a common feature of DDoS attacks) from arriving at their target. However, if a spoofed IP address were to fall inside the predefined range, it would not be blocked. In addition, not all DDoS attacks use spoofed IP addresses. Further, it could be the case that legitimate users are restricted from access to the network without some form of tunnelling. Therefore, this technique is not infallible. Other prevention techniques such as hop-count filtering (Jin et al., 2003), route-based packet filtering (Park and Lee, 2001), and history-based filtering (Peng et al., 2003), all suffer from similar drawbacks. Whilst being useful, they are not able to stop all DDoS attacks, particularly the more sophisticated ones.

Other means of DDoS attack prevention include the use of load balancing (NGINX, 2022) (Mahjabin et al., 2017). This is where the loads placed on the service host are spread out across sibling hosts. This helps prevent DDoS attacks as a much larger payload is required to disrupt the system - as more than one host is available for legitimate connections. However, this method requires a significant investment from the service owner as replica servers are required to be available at all times. Therefore, this is not an ideal solution for small to medium-sized enterprises or organisations. Other, more simple methods essentially require an awareness of the issue from the service owner. Ensuring unnecessary services are disabled and applying up-to-date security patches to host systems can go a long way in helping prevent DDoS attacks and other types of attacks. However, even a combination of all of the aforementioned methods cannot be relied upon to prevent all DDoS attacks.

It is, therefore, necessary to be able to respond to attacks that evade these measures and mitigate their effects. In order to respond to a DDoS attack effectively, it is crucial that the attack is detected as quickly as possible. As demonstrated in section 2, the problem of detecting DDoS attacks has been the subject of a lot of research in recent years. The increase in our use and reliance on connectivity has increased the attack surface for attackers to target and the issues spawned from this are clear – so the reason for this heightened attention in research is justified.

DDoS attack detection is a difficult problem to solve as it is challenging to distinguish between legitimate traffic and malicious traffic. In general, two categories of DDoS attack detection are most often considered: signature-based and anomaly-based detection.

Signature-based detection systems use a selection of known attack indicators to identify malicious traffic. These systems can be effective at preventing attacks with signatures in the detection system's database, however, they are ineffective at recognising slight variations of known attacks and cannot detect novel attacks whatsoever due to the open-set recognition problem.

Anomaly-based detection systems differ in that they simply monitor network traffic for outliers, as opposed to specific attacks, and use a particular function to determine if such traffic is potentially malicious. Naturally, anomaly-based detection systems require a baseline understanding of the network they are operating on in order to reliably detect outliers, this can take some time to establish as there are often large variations in network traffic for many external reasons, such as time of year and socioeconomic changes as well as sudden internal changes. Therefore, they can be less effective in the short term.

Section 2 shows that much of the modern research into DDoS attack detection uses machine learning models. The approaches used can be considered part of the above two categories. Supervised classifiers fall into the signature-based detection category, as the

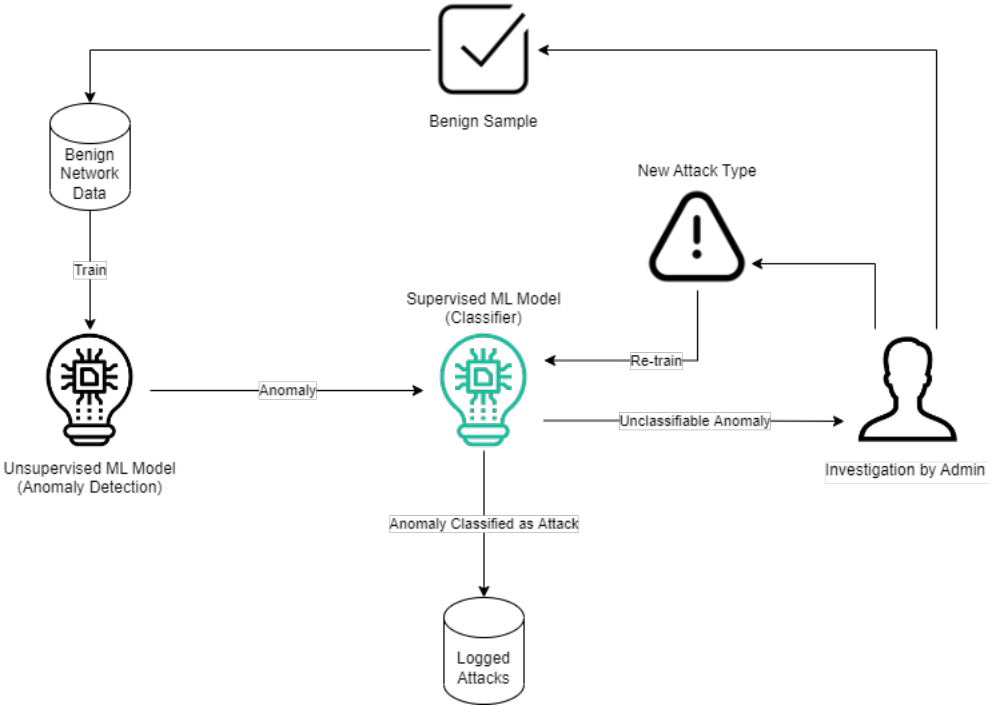


Figure 1: High-level diagram of the DDoS detection system

features used in classification can be considered signatures of the labelled attack. Unsupervised anomaly detection models naturally fall into the other category. The issues with using a single approach from these two categories have been discussed. It would seem a more reliable method of detecting DDoS attacks would be to use a two-pronged approach.

4 DDOS DETECTION SYSTEM

The detection system proposed in this work is a hybrid machine learning system made from a combination of an unsupervised anomaly detection module and a supervised classification module. The two modules work in tandem to provide a reliable detection method for known and novel DDoS attack types, whilst also informing the network owner of the nature of the attack. In addition, the system uses iterative learning to continuously improve its ability to detect and recognise known and novel DDoS attacks in a network. A high-level diagram of the overall system can be seen in Figure 1.

The unsupervised anomaly detection module operates based on the assumption that any traffic not recognised as benign could be part of a DDoS attack. It is trained on benign network traffic data and recognises outliers as potentially part of an attack. Any outliers are passed onto the classification module, this is a supervised machine learning model trained on data from historical DDoS attack data from real, known attacks. It outputs the probabilities of the outlying sample belonging to a particular class of DDoS attack from the set of known attack classes. If the maximum output probability is below a specified threshold, the sample can be considered to be unclassifiable. Any maximum probability over this threshold suggests the sample can be considered part of the corresponding attack and is reported accordingly. Unclassifiable outlying samples are passed to a human operator – in this example, a system administrator. The administrator can investigate the sample more thoroughly and determine if it belongs to an attack or if it is benign. The administrator labels the sample as such and the sample is passed back to the system. Benign samples are passed back to the unsupervised anomaly detection module to retrain and improve the model such that future, comparable samples are recognised correctly. Similarly, if the unclassifiable sample is determined to be part of a new or known attack type, the administrator can label it as such and the sample is passed back to the supervised classification module to retrain the model.

5 IMPLEMENTATION

The prototype DDoS detection system was implemented in Python as this is the programming language most familiar to the author, and to leverage its powerful data-science capabilities. Python has many well-supported libraries such as Pandas (NumFOCUS, 2022), Numpy (NumPy, 2022) and scikit-learn (scikit learn, 2022b) that deal with the problems posed by the development of this system in a clean fashion.

Due to time constraints and the necessity of reliability, the machine learning algorithms used in the implementation of the DDoS detection system made use of external libraries. Whilst the external library code will be more reliable than newly-written code, it does restrict which algorithms can be used to the set of algorithms that the library provides. The scikit-learn library was used to implement all machine learning algorithms used in this prototype system.

5.1 Machine Learning Algorithm for Anomaly Detection

The general idea of anomaly detection is to consider a set of n samples from a data set with m attributes, each sample can be described by those features and plotted in an m-dimensional space. If a new sample is added, we can plot this sample into the existing space and determine using a variety of techniques whether this sample is close enough to an existing cluster of samples to be considered part of the cluster, samples that are not close enough to the cluster are outliers.

As stated, the number of possible algorithms to choose from for the anomaly detection model was limited to those provided by the scikit-learn library (scikit learn, 2022a), these are One-class Support Vector Machine (SVM), Isolation Forest and Local Outlier Factor (LOF). All three were tested to find the most reliable for this system.

5.1.1 One-class SVM

Support Vector Machines are normally considered to be supervised learning models and are most often used for classification tasks. If we have a dataset where there are two classes: positive and negative, the regular SVM finds the maximum-margin hyperplane, that is, the hyperplane that is equidistant from the two sets of data points. This hyperplane is then used to classify future samples by determining which side of the hyperplane the sample lies on.

For a one-class SVM (Schölkopf et al., 1999), all samples in the dataset are considered to be part of the same class. We can plot the data points onto an m-dimensional space and consider these points to be part of the negative class. We then consider the origin to be the point of the positive class. This allows us to compute a hyperplane that is equidistant from the two class clusters, and to determine where new samples fall. Those that fall on the positive side of the hyperplane, are considered outliers. The one-class SVM uses the kernel trick (Shawe-Taylor and Cristianini, 2004) in particular, a Gaussian kernel (Schölkopf et al., 1995) to operate in a high-dimensional space and increase the chances of separability from the origin, allowing outliers in any direction to be placed in the negative space. The details of this particular technique are out of the scope of this report.

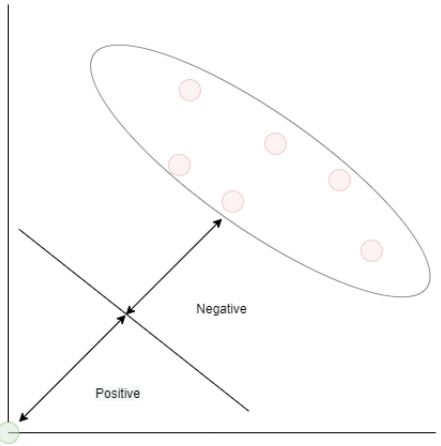


Figure 2: One-class SVM hyperplane split (not to scale)

5.1.2 Isolation Forest

An Isolation Forest (Liu et al., 2008) is a method of anomaly detection that uses a multitude of isolation trees to isolate each data point in a set by splitting them into separate partitions, each isolation tree is polled and an anomaly score is calculated. It works similarly to a Random Forest, discussed in 5.2.1, with the difference that there is a one-to-one mapping of partitions to data points, rather than a one-to-many mapping where all data points in a partition are homogeneous.

An isolation tree is built by randomly selecting an attribute from the data set and then randomly partitioning along the range of values for that attribute. This is repeated until all points are isolated into separate partitions. Recursive partitioning can be represented by a tree structure, thus the path length from the root node in a given tree t to the leaf node that represents a particular sample (or more specifically, the partition containing said sample) is the value v. Each isolation tree can be polled for the value v_t and the

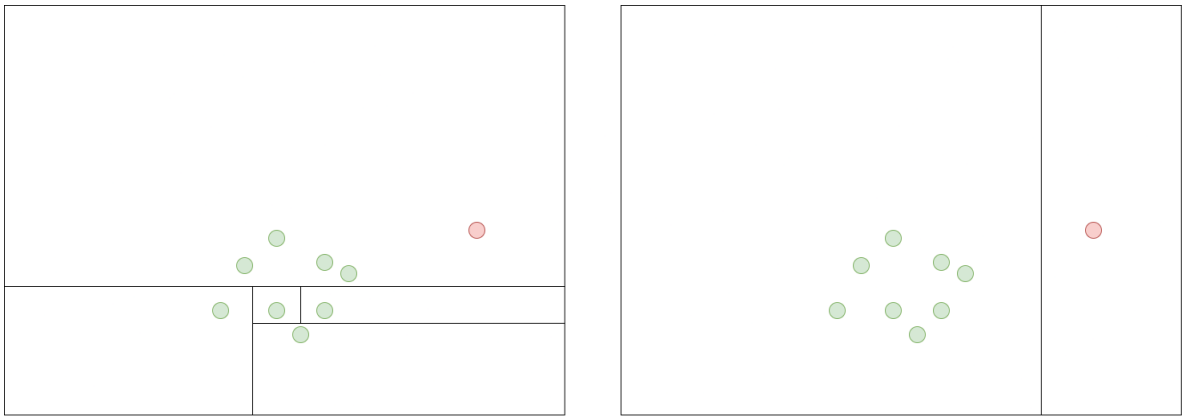


Figure 3: Isolation tree comparison for isolation of regular and anomalous samples

decision function can be calculated with the formula:

$$V = \frac{\sum_{t \in T} v_t}{|T|}$$

Where V is the final anomaly score of the sample and T is the set of isolation trees for the isolation forest. The isolation forest works on the intuition that anomalous samples will require fewer partitions to isolate, thus a low comparative score for a given sample indicates an outlier. Figure 3 shows an example where an anomalous sample is isolated in just 1 partition (*right*), compared to the 4 required to isolate the regular sample (*left*).

5.1.3 Local Outlier Factor (LOF)

The LOF algorithm (Breunig, M.M. and Kriegel, H.-P. and Ng, R.T. and Sanger, J., 2000) measures the degree of abnormality of a given data point by computing a score, known as the local outlier factor. This score is a measurement of the density deviation from a neighbouring cluster of data points for a given data point. The local outlier factor is based on the ratio of the average density of its k-nearest neighbours to its own density. The intuition is that a regular data point is expected to have a density similar to that of its neighbours, whilst an outlying sample is expected to have a much smaller density score.

5.1.4 Anomaly Detection Algorithm Performance Comparison

Each of the three algorithms was evaluated to determine the most reliable for the DDoS detection system. The data used to train each model was sourced from the CICDDoS2019 data set from the Canadian Institute for Cybersecurity (Canadian Institute for Cybersecurity, New Brunswick, 2019). This data set contains realistic network packet-flow data demonstrating the effect and signatures of numerous types of DDoS attacks on a network. It is split into separate comma-separated value (CSV) files each containing data pertaining to a different type of DDoS attack. The types of DDoS attack represented in the data set include SYN-flood, UDP-flood, UDP-Lag, and Portmap, LDAP, MS-SQL and NetBIOS reflective attacks. The data set also contains benign traffic data amongst the attack data. The UDP-Lag set was removed entirely from use in the creation of and testing of this system due to the lack of samples provided by the data set relative to other types of attacks.

The problem of anomaly detection is essentially a binary classification problem – does the new sample belong to the group, or is it an outlier? This allows us to disregard the type of DDoS attack at this stage, and simply label each sample as BENIGN (0) or ATTACK (1). This was done to all of the data as part of preprocessing. An equal amount of BENIGN samples were taken from each CSV file to get a range of data and avoid biases, these were selected at random to avoid the problem of data non-uniformity, and were combined to create a single superset of data to train the models with.

Each CSV file in the CICDDoS2019 data set contains 86 features and a class label. This is far too many for effective training of a machine learning model, and indeed too many features can cause issues with overfitting. Therefore, features were selected based on their correlation with the label and lack of correlation with each other, as this is a standard approach (Hall, 1999). The correlation to the label was limited to between 0.75 and 0.9, with the latter limit to reduce the chance of overfitting the data. Intuitively, features such as Source IP and Destination IP were removed with this process, as these would create a non-network-agnostic model. The Flow ID

Table 1: Selected features for the training of ML models

Feature	Description
Protocol	The network protocol used
Fwd Packet Length Min	Minimum size of packet in forward direction
Fwd Packet Length Mean	Average size of packet in forward direction
Bwd Packet Length Min	Minimum size of packet in backward direction
Min Packet Length	Minimum length of a packet
Packet Length Mean	Average length of a packet
ACK Flag Count	Number of packets with ACK flag
URG Flag Count	Number of packets with URG flag
Down/Up Ratio	Download and upload ratio
Average Packet Size	Average packet size (bytes)
Avg Fwd Segment Size	Average size observed in the forward direction

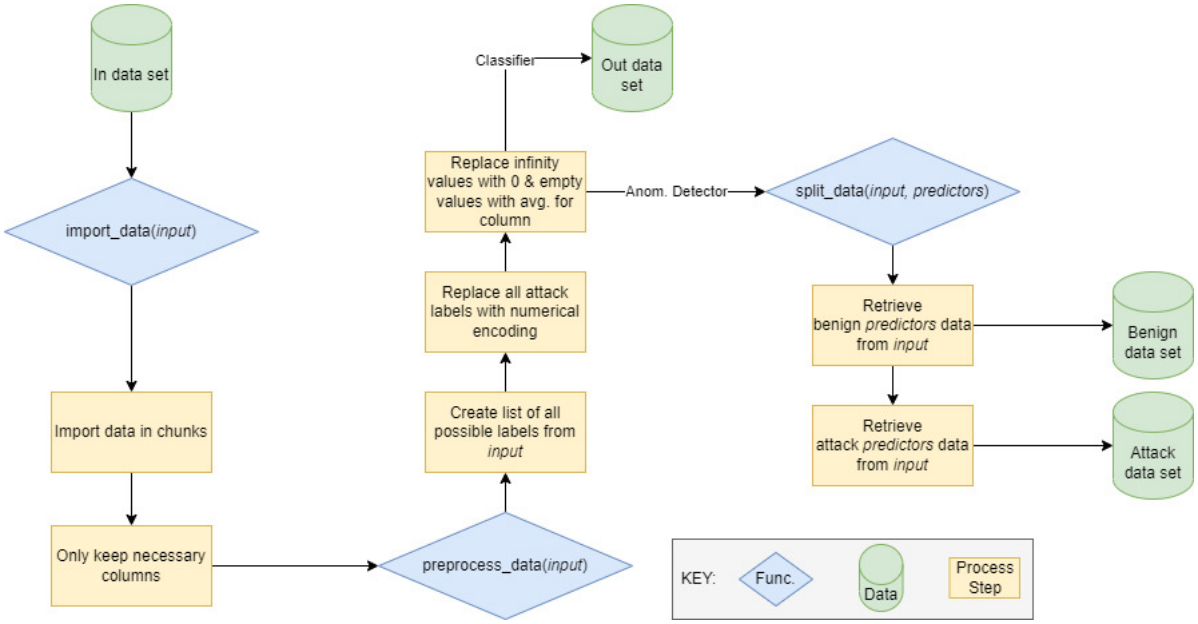


Figure 4: Data preprocessing steps

and Timestamp features were also omitted manually as these are irrelevant to future predictions. The final set of features that were used for the training and testing of the model are shown in Table 1 with a description in the context of a packet flow from an outside source to an inside host. Descriptions were obtained from the documentation for the CICFlowMeter software (Canadian Institute for Cybersecurity, New Brunswick, 2017a) used to extract the data from packet captures.

In addition to the above, it was necessary to preprocess the data to remove bad or unusable values - such as *infinity* and null values. These were replaced with 0 (*note: no column with infinity values also had legitimate 0 values*) and the mean of all other values in the column, respectively. Furthermore, all labels in the data were replaced with numerical encodings, these were simply integers starting from 0 for each label in the data, this is necessary as scikit-learn is unable to deal with non-numerical categorical values. For the anomaly detection data, it was only necessary to include samples with a BENIGN (or 0-encoded) label. However, ATTACK data was also kept separately for testing purposes. Data for the classifier included both BENIGN and ATTACK samples. A flow chart of the preprocessing script usage can be seen in Figure 4, the code used is included in the appendix as Preprocess.py.

The models trained using benign data were each tested using an equal amount of unseen BENIGN and ATTACK samples from each CSV file combined into a single file. The test aimed to measure the reliability of each algorithm in correctly identifying data that belonged to the group (negative - benign data) and out-

Table 2: Anomaly Detection Algorithm Comparison Results

	One-class SVM	**Isolation Forest**	**LOF**
Total Test Samples	203154	203154	203154
True Positives	183026	203055	203030
False Positives	47	10	20
True Negatives	17	54	44
False Negatives	20064	35	60
Accuracy	0.9010	0.9998	0.9996
Sensitivity	0.9012	0.9998	0.9997
Specificity	0.2656	0.8438	0.6875
Precision	0.9997	1	0.9999
F1 Score	0.9479	0.9999	0.9998

liers (positive - attack data). Test results can be seen in Table 2.

The first thing to note is that the number of regular (BENIGN) samples used in this test is far lower than the number of outlier (ATTACK) samples. This is because of the limits of the CICDDoS2019 data set. Each CSV file has a limited number of benign samples to train the model with, compared to attack samples. It was decided that the majority of these would be used to train the model to ensure it has the best chance of identifying outliers. Therefore, only a handful were kept back from the training set for testing. It is more important in this system that positive outliers are detected than negative samples missed, as a false alarm DDoS attack would be less damaging than completely missing a real attack. Furthermore, the nature of the system means that if a False Positive DDoS attack is detected by the anomaly detection module, it will be passed to the classification module which should filter it out as non-malicious.

From these results, it is clear that the best overall performer in this test is the Isolation Forest algorithm, it outperformed the other two algorithms in every measure. All had a high score for accuracy, however, this measure is biased due to the far larger amount of positive (ATTACK) samples in the data. The precision score links to this, as all performed very well in identifying positive samples, so accuracy is skewed by this and it was merely shown for completeness. Where the algorithms differed was mostly seen in the correct identification of negative (BENIGN) samples. Interestingly, there was a massive disparity between the One-class SVM model and the other two in this aspect, this model saw a huge amount of False Negative predictions and can therefore be considered unreliable for anomaly detection in this system – False Negatives here equate to missed attack samples, and there are far too many misses here for it to be relied upon. The other two models were much closer, however, the Isolation Forest model outperformed the LOF model slightly in identifying a greater proportion of negative and positive samples correctly, thus this algorithm was chosen for the anomaly detection module of the DDoS detection system.

5.2 Machine Learning for DDoS Attack Classification

Classification is the mapping of a set of input variables to a discrete set of output variables. It can be seen as the process of predicting the label of a particular data point such that it can be placed into a known category. Classification is used in the DDoS detection system to identify the type of DDoS attack that the anomaly detection module has detected.

The possible algorithms to choose from for this part of the system were again limited by those provided by the scikit-learn library (scikit learn, 2022d). In this respect, there are more provided than with anomaly detection. However, due to time constraints, not all could be tested for reliability. It was therefore decided that only a select few would be tested. These were Random Forest, Logistic Regression, *k*-Nearest Neighbour (*k*-NN), and Multilayer Perceptron (MLP). Each was mentioned in the literature in 2 and was shown to be either the strongest or the second strongest algorithm for its research case. As these are more well-known algorithms than those used for anomaly detection, only a high-level description of each will be provided here for brevity.

5.2.1 Random Forest

The Random Forest algorithm (Ho, 1995) is an ensemble method of learning that can be used for classification tasks. Akin to the Isolation Forest described in 5.1.2, it makes use of a collection of Decision Trees that are polled to provide a more reliable prediction.

5.2.2 Logistic Regression

Logistic Regression uses the logistic function to model the probability that a sample belongs to a particular class, this is usually a binary decision as discussed in 2. In this case, Logistic Regression is expanded to a Multinomial Logistic Regression to allow for more than two classes to be predicted. Scikit-learn compares the probability values of each class (scikit learn, 2022c) in order to determine the most likely class for a given sample.

5.2.3 *k*-Nearest Neighbour (*k*-NN)

The *k*-Nearest Neighbour algorithm (Fix and Hodges, 1951) is trained by generating a vector in *n*-dimensional space (where *n* is the number of predictors for the sample) for every training sample, each given a class label. Predictions are then made by generating a further vector for each new sample and comparing the distance (usually Euclidean distance) to the nearest *k* vectors; the class label of the majority of these *k* vectors is assigned as the predicted class label for the new sample.

5.2.4 Multilayer Perceptron (MLP)

The Multilayer Perceptron algorithm is a neural network model that learns a function from the training data set sample, that is it creates a mapping from input (the predictor values) to output (the class label) using backpropagation. The model works by creating three or more layers of neurons: one input, at least one "hidden" and one output. Each neuron in each set of "hidden" layers in the Multilayer Perceptron has a function that maps to every node in the following layer until the output layer is reached, this can then be reduced to a direct input-output mapping.

5.2.5 Classification Algorithm Performance Comparison

Whilst the same data set was used, testing each classification algorithm involved a slightly different process to the testing of the anomaly detection algorithms in 5.1.4. Firstly, due to it being a multinomial classification problem, analysis of predictions for each possible class is necessary. Secondly, our system requires that unknown classes are recognised as such – these are the novel DDoS attacks that are recognised in anomalous traffic by the anomaly detection module but cannot be classified by the classification module.

To determine if a sample is unclassifiable, we need to define a boundary probability value, where if no class probability value for a given sample exceeds the boundary value, we consider that sample to be unclassifiable. Therefore, we do not take the final prediction output from the classifier, but an array of probabilities corresponding to each possible class. The probability boundary value was set to 0.9 for the purposes of testing.

Each algorithm model was trained using a superset of equal amounts of benign and malicious traffic from each CSV file in the data set *except* the Portmap data, with each CSV file randomly sampled to ensure uniformity. The test superset was created from samples not used in the training superset, including samples from the Portmap data. The aim of the test was to determine the reliability of classification for all known types of DDoS attacks and to determine the reliability of recognising the unknown (Portmap) as *unclassifiable*. The results of these tests can be seen in Tables 3, 4, 5 & 6. Note that the results were taken directly from the built-in scikit-learn classification report to reduce human error and to save time, thus the formatting and values shown here differ from the manually calculated results shown in 5.1.4, but should be sufficient for comparison.

From these results, we can note that all algorithms underperformed in the reliability of determining the unclassifiable attack type compared to all other classes, in almost all cases. This can be attributed to the boundary probability value and there is a chance this could be improved with calibration. For the purposes of comparison, this is not important as the boundary value was kept consistent throughout the experiment.

The *k*-NN algorithm generally has a very strong accuracy in predicting known attack types correctly. However, overall, the results show the Random Forest algorithm to be the most reliable in predicting a DDoS attack class when the unclassifiable attack type is also considered. The Random Forest model will be used for the classification module in the DDoS detection system.

5.3 System Prototype

As discussed, the prototype system was implemented in Python. It is made up of three Python modules: *Anomaly.py*, *Classification.py*, and *Detection.py*. Naturally, the first two represent the anomaly detection and classification modules of the system. The third contains functionality to connect the first two and produce an output. Each will be described in this section with truncated Python code (omitting minor technicalities). The actual Python files are included in the attached appendix files of the project.

Table 3: Random Forest Test Results

Random Forest							
	Benign	**NetBIOS**	**LDAP**	**MSSQL**	**SYN-Flood**	**UDP-Flood**	**Unclassifiable (Portmap)**
Total Samples	86694						
Sensitivity	1.00	1.00	1.00	0.98	1.00	0.98	0.80
Precision	1.00	0.98	0.98	0.84	1.00	0.98	0.98
F1 Score	1.00	1.00	0.99	0.98	1.00	0.99	0.88

Table 4: Logistic Regression Test Results

Logistic Regression							
	Benign	**NetBIOS**	**LDAP**	**MSSQL**	**SYN-Flood**	**UDP-Flood**	**Unclassifiable (Portmap)**
Total Samples	86694						
Sensitivity	0.63	1.00	1.00	0.94	1.00	0.51	0.78
Precision	1.00	0.99	0.94	0.66	0.73	0.98	0.84
F1 Score	0.77	1.00	0.97	0.77	0.84	0.67	0.81

Table 5: *k*-NN Test Results

***k*-Nearest Neighbour**							
	Benign	**NetBIOS**	**LDAP**	**MSSQL**	**SYN-Flood**	**UDP-Flood**	**Unclassifiable (Portmap)**
Total Samples	86694						
Sensitivity	1.00	1.00	1.00	0.97	1.00	0.98	0.79
Precision	1.00	1.00	0.98	0.98	1.00	0.99	0.89
F1 Score	1.00	1.00	0.99	0.98	1.00	0.99	0.84

Table 6: Multilayer Perceptron Test Results

Multilayer Perceptron							
	Benign	**NetBIOS**	**LDAP**	**MSSQL**	**SYN-Flood**	**UDP-Flood**	**Unclassifiable (Portmap)**
Total Samples	86694						
Sensitivity	1.00	1.00	0.99	0.89	1.00	0.98	0.80
Precision	1.00	1.00	0.96	0.97	1.00	0.93	0.95
F1 Score	1.00	1.00	0.98	0.93	1.00	0.95	0.87

5.3.1 Anomaly.py

The pseudo-Python code for this module can be seen in Code 1.

Line 6 shows the *anom_prediction* function which takes a sample as input and returns a prediction of either 1 for regular samples or -1 for anomalous samples. It first converts the input data to a Pandas dataframe for easier processing, it then extracts only the predictor values that it requires and converts them to a Numpy array. The model is loaded using the pickle module (an object serialisation library for saving and loading the model) (Python, 2022) and the Numpy array of values is used as input to the model's prediction function, which returns the prediction.

The following function, train_model, is a method used for the iterative learning functionality of the system. This takes a new input value and appends it to the existing training data set for future use, then retrains the model using the new data set. Note that in this function, the right-facing arrows indicate saving the input (left) to disk at the location defined by the variable (right).

5.3.2 Classification.py

The pseudo-Python code for this module can be seen in Code 2.

Similar to *Anomaly.py*, this module has two functions. The first, described from line 6, is a method for retrieving the probabilities of a given sample belonging to each known class. This works in much the same way as the anom_prediction function in Code 1, but instead of returning a 1 or -1, it returns a list of prob-

Code 1 Anomaly.py

```
1: import pandas, sklearn, pickle
2: MODEL = < path to anom_model > .sav
3: MODEL_PREDICTORS = [< set of predictors >]
4: TRAINING_SET = < path to benign_set > .csv
5:
6: function anom_prediction(input)
7:     df = pandas.DataFrame(input).[MODEL_PREDICTORS]
8:     vals = df.to_numpy()
9:     loaded_model = pickle.load(MODEL)
10:    prediction = model.predict(vals)
11:    return prediction
12: end function
13:
14: function train_model(input)
15:    df = pandas.DataFrame(input)
16:    tset = pandas.read_csv(TRAINING_SET)
17:    tset.append(df)
18:    pandas.to_csv(tset) → TRAINING_SET
19:    new_model = sklearn.IsolationForest()
20:    new_model.fit(tset)
21:    pickle.save(new_model) → MODEL
22: end function
```

Code 2 Classification.py

```
1: import pandas, sklearn, pickle
2: MODEL = < path to class_model > .sav
3: MODEL_PREDICTORS = [< set of predictors >]
4: TRAINING_SET = < path to attack_set > .csv
5:
6: function class_probabs(input)
7:     df = pandas.DataFrame(input).[MODEL_PREDICTORS]
8:     vals = df.to_numpy()
9:     loaded_model = pickle.load(MODEL)
10:    probs = model.predict_proba(vals)
11:    return probs
12: end function
13:
14: function train_model(input)
15:    df = pandas.DataFrame(input)
16:    tset = pandas.read_csv(TRAINING_SET)
17:    tset.append(df)
18:    pandas.to_csv(tset) → TRAINING_SET
19:    new_model = sklearn.RandomForest()
20:    new_model.fit(tset)
21:    pickle.save(new_model) → MODEL
22: end function
```

abilities. Again, the train_model function described from line 14 is a method of retraining the model with new inputs from the iterative learning process, this differs here only in that a Random Forest model is used.

Code 3 Detection.py

```
1:  from Anomaly import anom_prediction
2:  from Classification import class_probabs
3:
4:  function run(input)
5:      if anom_prediction(input) == −1 then
6:          probabs = class_probabs(input)
7:          if max(probabs) >= prob_threshold then
8:              return probabs[index(max(probabs))]
9:          else
10:             return −1
11:         end if
12:     else
13:         return 0
14:     end if
15: end function
```

5.3.3 Detection.py

The pseudo-Python code for this module can be seen in Code 3.

This module is used to integrate the anomaly detection and classification modules into one cohesive system. It uses a very straightforward algorithm in the run function, shown from line 4. It first executes the anom_prediction function from the *Anomaly.py* module which returns a -1 if the input sample is anomalous, and 1 if not. If the sample is deemed benign, the function returns a 0 to signify this. If the sample is deemed anomalous, it is passed to the class_probabs function from the *Classification.py* module which attempts to classify the sample. Again, this function returns a list of probabilities corresponding to each possible known class of DDoS attack.

If the maximum probability k in the list is *greater* than the probability threshold, then the index of k in the list is taken as the class of attack. This works because each DDoS attack type is encoded with an ordinal value ($0, 1, 2 \ldots n$) due to the inability of scikit-learn to deal with categorical values. Furthermore, lists in Python are ordered. Therefore, so long as we keep the position of each probability in the probability list the same, we can safely say that the max probability index corresponds to the encoded value of the attack type. If k is *less* than the probability threshold, the classifier is not certain of the attack type and returns -1 to signify this. The sample can then be further investigated by a human operator to determine if it is benign or a new type of attack.

6 TESTING METHODOLOGY & RESULTS

6.1 Network Topology

To properly test the reliability and portability of this prototype DDoS detection system, it was necessary to collect new data from a fresh source. This could have been collected from existing, publicly available data different from the CICDDoS2019 set. However, the issue with this is the inconsistency of the formatting of the data. The CICDDoS2019 set contains packet flow data from packet capture (pcap) files collected using Wireshark (Wireshark, 2022). As mentioned, the packet flow data is extracted using the CICFlowMeter software (Canadian Institute for Cybersecurity, New Brunswick, 2017a), which outputs the data as a CSV file. This is not a consistent method of formatting throughout all DDoS attack data sets, therefore we cannot test a model trained on data from CICDDoS2019 with those data sets.

A better method would be to use Wireshark (Wireshark, 2022) to capture packet data from a simulated attack and extract packet flow data in the same format as CICDDoS2019 using the CICFlowMeter tool. This was made possible using Cardiff University's Cyber Range. The Cyber Range allows for the virtualisation of a relatively large amount of machines and connections between them, thus enabling the creation of virtualised network topologies. It is, therefore, possible to create a simulated network environment that models an organisation and to perform a DDoS attack on this network using further virtualised machines. This was the method chosen to test the prototype DDoS detection system, and the implementation details of this

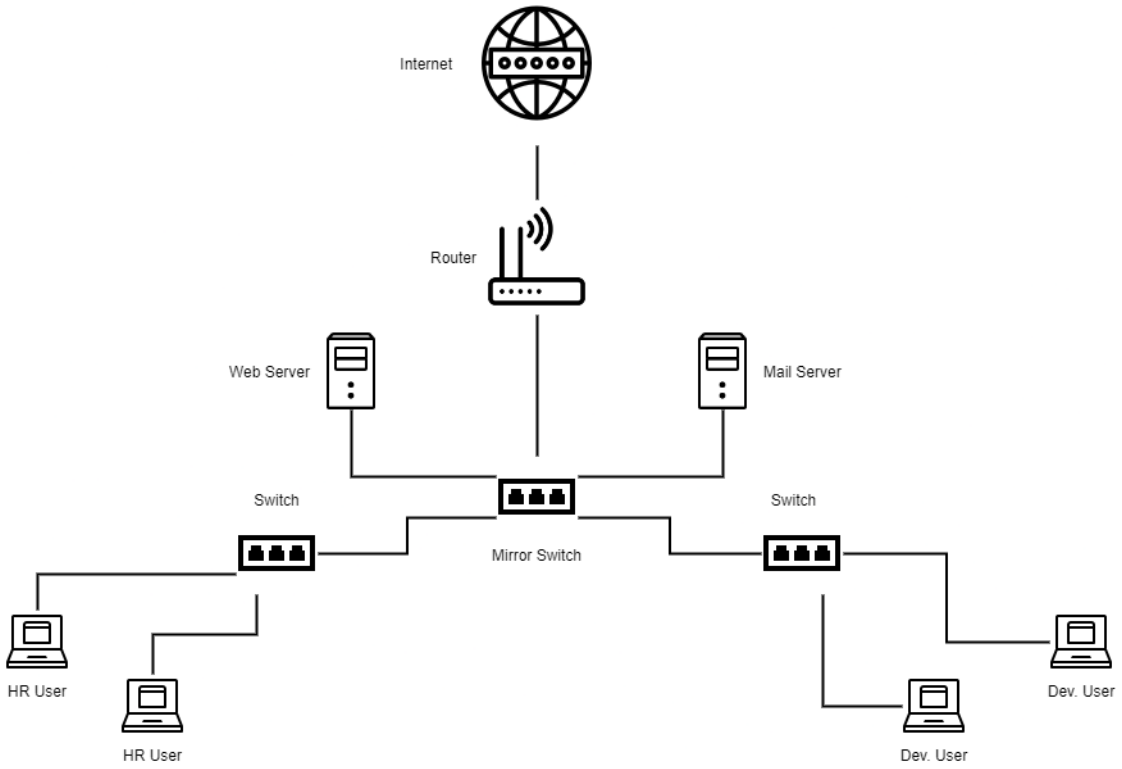


Figure 5: Small business star topology network

test follow.

One can imagine a small business workplace comprised of two or three rooms with machines connected to a switch in each room, all connected to a central router that acts as a gateway to the Internet. The business may also have a web server and Email server. This would be a fairly typical star topology, and an illustration of such a network can be seen in Figure 5.

Ultimately, this was the type of network emulated on the Cyber Range for our test. The beauty of a network like this is that its centralised nature ensures that (excepting certain direct connections between end-user machines, which can be omitted) all packets pass through the same point, in this case, the central switch. This allows us to easily collect all packet data that passes through the network by tapping into this location and collecting the packets. We do this using a type of switch known as a mirror switch. The mirror switch has multiple regular ports that work as normal, whilst also having a mirror port where a copy of every packet transmitted through all other ports is sent.

6.2 Implementation on the Cyber Range

The Cyber Range implementation of the topology described in 6.1 can be seen in Figure 6. Firstly, the implementation differs from the originally described topology in that there is no connection to the Internet. This is partly due to technological restrictions but also because such an implementation was not necessary to perform a test. The topology also has a host for capturing the packet data, connected directly to the mirror switch, and two attacking hosts connected indirectly via one further switch. In addition, the implementation has a DNS server to aid in routing to the mail server. The interface to the Cyber Range is the DIATEAM Hyneview software (DIATEAM, 2022).

The user systems are virtualised Ubuntu Linux machines. Each is labelled with a randomly generated user name to distinguish between them, e.g. “z.dudley”. Each user host can perform basic actions such as email, creating files, and accessing the web server. The use of these is further described in 6.3. They are represented graphically in the topology by the computer monitor.

The three servers in the topology include the Email server, a DNS and a simple web server. The first two are implemented using Windows Server, with the Email server utilising the open-source hmail software (hMailServer, 2022) and the DNS server using the Windows DNS Server Manager (Microsoft, 2021). The web server is a simple Apache (The

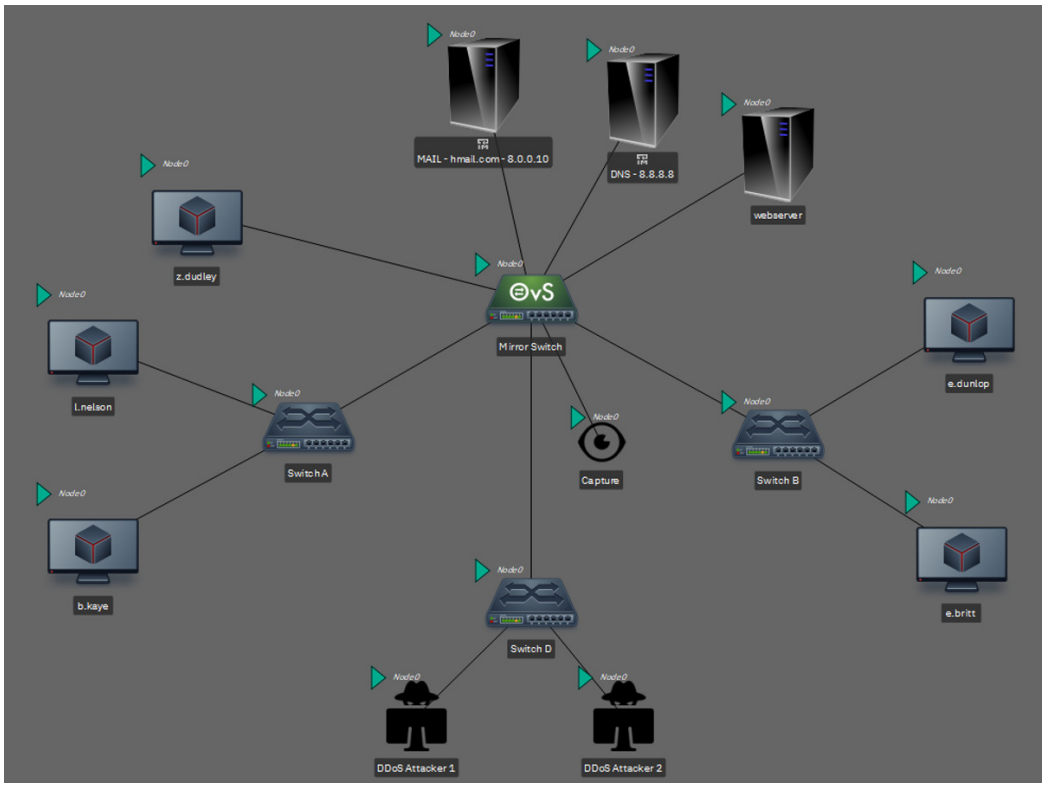


Figure 6: Functional Cyber Range topology

Apache Software Foundation, 2022) server hosted on an Ubuntu Linux machine. These are represented graphically in the topology by the black tower machines.

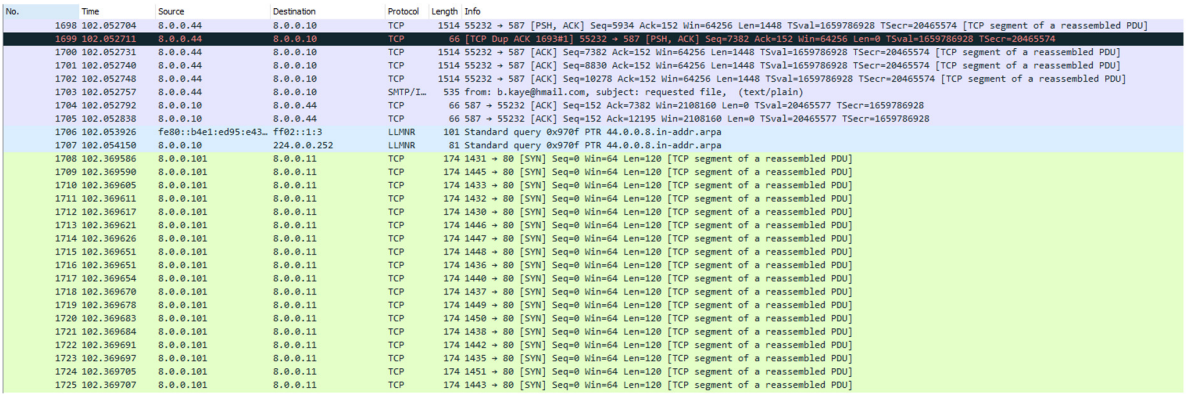
The DDoS attacker hosts are Kali Linux (Off-Sec Services Limited, 2022b) virtual machines represented by the hacker icon in the topology. The function of these is discussed in detail in 6.4. Packet capturing using the Cyber Range can be done using the Hyneview software directly. However, to capture from a mirror switch, the mirror port must have a connection to a host, so we have a dummy-host labelled Capture represented graphically by the eye icon in the topology for this purpose. Finally, there are four switches, with the most central one being the mirror switch.

6.3 Simulating Legitimate Activity

Naturally, a normal network is created for legitimate users, DDoS attacks are a phenomenon that occurs as a result of this legitimate use – if a network was not used, there would be no need to attack it. This means that any traffic associated with a DDoS attack on a network will almost certainly be present alongside legitimate traffic. One of the main difficulties encountered when trying to implement a DDoS detection system is to ensure it can reliably distinguish between legitimate and malicious traffic. To ensure that our system can do this, we need to simulate legitimate use as well as malicious use.

In our topology, we have a set of hosts belonging to users. The users at this simulated organisation are able to perform some basic actions. To simulate realistic usage, and to provide constant activity on the network, a simple Python script was written that is present on all user hosts. The details of this script will not be described here, although the script is included in the appendix as *SimUser.py*. Suffice to say the script performs a user-defined number of the following actions at random:

- Generate a text file – generates a simple text file with a random string of text
- Send an email – sends an email to another named user on the network, with a generated file attached
- Access website – accesses the website hosted by the web server and downloads the page
- Pause – pauses for a random amount of time up to 30 seconds to better simulate realistic use



No.	Time	Source	Destination	Protocol	Length	Info
1698	102.052704	8.0.0.44	8.0.0.10	TCP	1514	55232 → 587 [PSH, ACK] Seq=5934 Ack=152 Win=64256 Len=1448 TSval=1659786928 TSecr=20465574 [TCP segment of a reassembled PDU]
1699	102.052711	8.0.0.44	8.0.0.10	TCP	66	[TCP Dup ACK 1693#1] 55232 → 587 [PSH, ACK] Seq=7382 Ack=152 Win=64256 Len=0 TSval=1659786928 TSecr=20465574
1700	102.052731	8.0.0.44	8.0.0.10	TCP	1514	55232 → 587 [ACK] Seq=7382 Ack=152 Win=64256 Len=1448 TSval=1659786928 TSecr=20465574 [TCP segment of a reassembled PDU]
1701	102.052740	8.0.0.44	8.0.0.10	TCP	1514	55232 → 587 [ACK] Seq=8830 Ack=152 Win=64256 Len=1448 TSval=1659786928 TSecr=20465574 [TCP segment of a reassembled PDU]
1702	102.052748	8.0.0.44	8.0.0.10	TCP	1514	55232 → 587 [ACK] Seq=10278 Ack=152 Win=64256 Len=1448 TSval=1659786928 TSecr=20465574 [TCP segment of a reassembled PDU]
1703	102.052757	8.0.0.44	8.0.0.10	SMTP/I…	535	from: b.kaye@hmail.com, subject: requested file, (text/plain)
1704	102.052792	8.0.0.10	8.0.0.44	TCP	66	587 → 55232 [ACK] Seq=152 Ack=7382 Win=2108160 Len=0 TSval=20465577 TSecr=1659786928
1705	102.052838	8.0.0.10	8.0.0.44	TCP	66	587 → 55232 [ACK] Seq=152 Ack=12195 Win=2108160 Len=0 TSval=20465577 TSecr=1659786928
1706	102.053926	fe80::b4e1:ed95:e43…	ff02::1:3	LLMNR	101	Standard query 0x970f PTR 44.0.0.8.in-addr.arpa
1707	102.054150	8.0.0.10	224.0.0.252	LLMNR	81	Standard query 0x970f PTR 44.0.0.8.in-addr.arpa
1708	102.369586	8.0.0.101	8.0.0.11	TCP	174	1431 → 80 [SYN] Seq=0 Win=64 Len=120 [TCP segment of a reassembled PDU]
1709	102.369590	8.0.0.101	8.0.0.11	TCP	174	1445 → 80 [SYN] Seq=0 Win=64 Len=120 [TCP segment of a reassembled PDU]
1710	102.369605	8.0.0.101	8.0.0.11	TCP	174	1433 → 80 [SYN] Seq=0 Win=64 Len=120 [TCP segment of a reassembled PDU]
1711	102.369611	8.0.0.101	8.0.0.11	TCP	174	1432 → 80 [SYN] Seq=0 Win=64 Len=120 [TCP segment of a reassembled PDU]
1712	102.369617	8.0.0.101	8.0.0.11	TCP	174	1430 → 80 [SYN] Seq=0 Win=64 Len=120 [TCP segment of a reassembled PDU]
1713	102.369621	8.0.0.101	8.0.0.11	TCP	174	1446 → 80 [SYN] Seq=0 Win=64 Len=120 [TCP segment of a reassembled PDU]
1714	102.369626	8.0.0.101	8.0.0.11	TCP	174	1447 → 80 [SYN] Seq=0 Win=64 Len=120 [TCP segment of a reassembled PDU]
1715	102.369651	8.0.0.101	8.0.0.11	TCP	174	1448 → 80 [SYN] Seq=0 Win=64 Len=120 [TCP segment of a reassembled PDU]
1716	102.369651	8.0.0.101	8.0.0.11	TCP	174	1436 → 80 [SYN] Seq=0 Win=64 Len=120 [TCP segment of a reassembled PDU]
1717	102.369654	8.0.0.101	8.0.0.11	TCP	174	1440 → 80 [SYN] Seq=0 Win=64 Len=120 [TCP segment of a reassembled PDU]
1718	102.369670	8.0.0.101	8.0.0.11	TCP	174	1437 → 80 [SYN] Seq=0 Win=64 Len=120 [TCP segment of a reassembled PDU]
1719	102.369678	8.0.0.101	8.0.0.11	TCP	174	1449 → 80 [SYN] Seq=0 Win=64 Len=120 [TCP segment of a reassembled PDU]
1720	102.369683	8.0.0.101	8.0.0.11	TCP	174	1450 → 80 [SYN] Seq=0 Win=64 Len=120 [TCP segment of a reassembled PDU]
1721	102.369684	8.0.0.101	8.0.0.11	TCP	174	1438 → 80 [SYN] Seq=0 Win=64 Len=120 [TCP segment of a reassembled PDU]
1722	102.369691	8.0.0.101	8.0.0.11	TCP	174	1442 → 80 [SYN] Seq=0 Win=64 Len=120 [TCP segment of a reassembled PDU]
1723	102.369697	8.0.0.101	8.0.0.11	TCP	174	1435 → 80 [SYN] Seq=0 Win=64 Len=120 [TCP segment of a reassembled PDU]
1724	102.369705	8.0.0.101	8.0.0.11	TCP	174	1451 → 80 [SYN] Seq=0 Win=64 Len=120 [TCP segment of a reassembled PDU]
1725	102.369707	8.0.0.101	8.0.0.11	TCP	174	1443 → 80 [SYN] Seq=0 Win=64 Len=120 [TCP segment of a reassembled PDU]

Figure 7: Packet capture data at the point of DDoS attack

6.4 Simulating a DDoS Attack

For this experiment, a DDoS attack was conducted against the web server. The web server was chosen as a target over the Email server due to the relative simplicity of testing whether or not the attack had been successful and because such attacks are more likely to occur in real scenarios than attacks against private Email servers due to the open-access nature of websites.

The type of DDoS attack chosen for this experiment was a SYN-flood attack. Performing such an attack is a simple task using the hping3 tool (OffSec Services Limited, 2022a) provided as standard with Kali Linux installations. This type of attack has associated packet flow data in the CICDDoS2019 data set, thus we can test the system's ability to detect SYN-flood attacks using completely new data – that is to say, in theory, there is no further training necessary for the system's machine learning models.

The following hping3 command was used to perform the attack:

hping3 -d 120 -S -p 80 –flood 8.0.0.11

To break this down:

- The -d option specifies the size of the packet data, in this case, 120 bytes
- The -S option tells the tool that we want all packets to be SYN packets
- The -p option specifies the destination port, in this case, port 80 was used as this is the HTTP port
- The -flood option tells the tool to send packets as fast as possible, thus flooding the target host
- The final argument is the destination IP address, 8.0.0.11, which is the IP of the web server on this network

Note that DDoS attacks are often conducted using spoofed source IP addresses, this was not the case for this experiment as this gave us a method of clear differentiation between malicious and benign packet data.

6.5 Test Steps & Results

The test was performed by first initiating the user-simulation script described in 6.3 for all five user hosts on the topology; each was initialised with 500 random actions to perform to ensure that none would stop whilst data was collected. Network traffic was checked using the built-in packet capture feature in Hyneview (akin to Wireshark) – it was confirmed that traffic was present on the network. Further, the connection to the website hosted by the web server was checked and it was confirmed that the page loaded properly.

At this point, the packet capture began. As mentioned, all traffic passes through the central switch in the topology. Therefore, the packet capture took place on the mirror port of this switch. The packet capture was set to stop at 400,000 packets as this was deemed a large enough number to provide a useful amount of flow data, but not too large as to make the packet capture file size unmanageable. The tool was left alone for a few minutes to capture a usable amount of benign data before performing the DDoS attack.

The DDoS attack was performed by executing the command described in 6.4 on the two attacker hosts in the topology. The specifications of the web server were very modest, and no DDoS prevention security was in place, thus only a small amount of attacking hosts were required to trigger a denial of service. Indeed, denial of service was confirmed almost instantly after the attack began. This was checked by attempting to access the website from a separate host machine – the page failed to load. Malicious packets were captured along with benign packets until the 400,000 packet limit was reached.

Table 7: Confusion matrix for detection of anomalous samples

	Predicted Positive	**Predicted Negative**
Actual Positive	70084	0
Actual Negative	32	141

Table 8: Confusion matrix for detection of SYN-flood attack samples

	Predicted Positive	**Predicted Negative**
Actual Positive	42337	27747
Actual Negative	0	173

Table 9: DDoS Detection System test results

	Benign	**SYN-Flood**	**Unclassifiable***
Total Samples	70257		
Predicted	141	42337	27779
Actual	173	70084	0
Sensitivity	1.00	0.6041	-
Precision	0.9995	1.00	-
Specificity	0.8150	1.00	-
F1-Score	0.9998	0.7532	-

Subsequently, the packet capture data was downloaded and checked in Wireshark for correctness, the data is shown in Figure 7. The figure shows the exact moment when SYN-flood attack packets began to be transmitted (green packets).

The raw packet data was then used as input into the CICFlowMeter tool which extracted data from 70,257 packet flows and stored the data in a structured CSV file comprising 173 *benign* flows and 70,084 *malicious* flows. The data required manual labelling, which was a trivial task as all data that originated from a malicious host (8.0.0.100 or 8.0.0.101) could be labelled as "Syn" whilst all other data could be labelled as "BENIGN". This data was then ready to use as input to the DDoS detection system, after preprocessing.

The models derived from the algorithm tests performed in 5 for the chosen algorithms (Isolation Forest and Random Forest) were used for testing with this new packet flow data. The test was done in this way because it allows the evaluation of the DDoS detection system's ability to be ported to new topologies and still be useful without requiring further training data.

The results of the DDoS detection system test using this new data are in tables 7, 8 & 9.

**Unclassifiable results in Table 9 only show the predicted count. This is because there were no samples in the testing data that should have been considered unclassifiable (types of attacks that the classifier had no prior knowledge of) – however, the system deemed some data to be so.*

Table 7 shows the confusion matrix for the detection of anomalous samples with the packet flow data from the Cyber Range test. We can see a relatively strong performance here in the detection system's ability to accurately distinguish between benign and malicious data, especially considering the system has not been trained with data from the simulation. These results suggest that the anomaly detection module works well, with an accuracy of 99.95%. The key point to take from this particular set of results is that it shows the system's ability to recognise when a DDoS attack has occurred – this is the most important part of any DDoS detection system, so these initial results are encouraging here.

Table 8 shows the confusion matrix for the correct identification of SYN-flood attack samples on the packet flow data from the Cyber Range test. This had a weaker performance compared to the results for simply detecting anomalous samples. With an accuracy of just 60.51%, we can suggest that the system does not perform especially well when classifying particular DDoS attacks, albeit with only the initial training.

One interesting point to note here is that the number of True Negatives shown in Table 8 is 173, the exact number of benign samples in the data, but Table 7 showed that errors were made, with 32 False Positives. This shows that whilst some benign traffic was initially mislabelled as *potentially malicious* by the anomaly detection module, the classification module deemed the same samples to be unclassifiable – indeed, Table 9 confirms this with the *Predicted* count of the $Unclassifiable$ samples. This is an excellent

Table 10: Confusion matrix for detection of anomalous samples with retrained model

	Predicted Positive	**Predicted Negative**
Actual Positive	56211	0
Actual Negative	2	155

Table 11: Confusion matrix for detection of SYN-flood attack samples with retrained model

	Predicted Positive	**Predicted Negative**
Actual Positive	56128	83
Actual Negative	0	157

Table 12: DDoS Detection System test results with retrained model

	Benign	**SYN-Flood**	**Unclassifiable***
Total Samples	56368		
Predicted	155	56128	85
Actual	157	56211	0
Sensitivity	1.00	0.9985	-
Precision	1.00	1.00	-
Specificity	0.9873	1.00	-
F1-Score	1.00	0.9993	-

outcome. The iterative learning process allows these *unclassifiable* samples to be later recognised as benign by a human administrator and fed back into training the anomaly detection module to ensure improved recognition. If these samples had been recognised as actual attacks after being flagged as anomalous, this opportunity may have been missed. It also means that no false alarms were triggered in this test – no benign data was ever considered part of a *definite* attack.

The weakness of the ability of the classification module to recognise the type of attack is a slight detraction. That said, considering that these are tests using models trained on a completely separate set of data, the overall results are positive. As discussed, all malicious samples were recognised as anomalous by the anomaly detection module, the classification of an attack as a particular type is less important than this. A network that uses a DDoS detection system based on this architecture could be configured to respond to *all* anomalous traffic in some way, regardless of its eventual classification – at the least, such outcomes could alert the possibility of an attack to the administrator. Furthermore, the iterative learning process allows for improvement with time. A legitimate attack that is recognised as anomalous at first, but later recognised as *unclassifiable* can be used to train the classification module, whilst false alarms can be used to train the anomaly detection module.

Following the collection of the first set of results, the system was retrained using unclassifiable data from the same test. To retest the system without the need to run a new simulation, only half of the unclassifiable samples were taken for the training phase, with the benign samples used to retrain the anomaly detection model, and the benign *and* malicious samples used to retrain the classifier. This equated to 16 benign samples and 13,873 malicious samples. These were then omitted from the test phase data set and the system was tested once again. Thus for this second, post-retraining test, we have 157 benign samples and 56,211 malicious samples, summing to 56,368 total samples. The results from the test on the newly-trained DDoS detection system can be seen in Tables 10, 11 & 12.

We can see a large improvement in reliability in these results. Here there are 2 False Positives for the predictions made by the anomaly detection module, these are initial false alarms, but again, they were recognised as unclassifiable by the classification module (as shown by the number of unclassifiable values in Table 12) thus were never registered as definite attacks. We also see a massive decrease in the number of False Negatives made by the classifier, with only 83 compared to the previous 27,747. Thus we have a massive improvement in the sensitivity and F1 scores from these tests. Furthermore, the number of anomalous samples correctly identified increased, with an improved specificity of 0.99 from 0.82. These results show the benefit of the iterative learning process and how this method can be used to quickly improve the system in a dramatic way.

7 CONCLUSIONS & FUTURE WORK

Whilst the work done during this investigation should be considered only an initial step, the findings suggest that this could be an interesting and successful pathway for denial-of-service prevention research. The DDoS detection system developed here is capable of dealing with issues that other methods struggle with, the chief of which is the capacity to detect new attacks. Initial results from tests performed in this investigation show that anomaly detection is a reliable method of detecting an attack regardless of type, and the logic involved in this is sound. Furthermore, the architecture of the system allows for continuous improvement by iterative learning, reducing any remaining error with time.

Although anomaly detection can be used to detect when a potential attack is taking place, it cannot say for sure that it *is* an attack. This system compensates for this problem by using a classifier in tandem with the anomaly detector. This is mostly a convenience for the network administrator or intrusion response system, as an attack of a known type can be mitigated faster and more reliably. However, if a sample can be reliably classified in this way, it can be said with more certainty that an attack is definitely happening and is not a false alarm. Further, the classifier enables iterative learning by ensuring that it has a high level of certainty before classifying an attack. This means that any time it is not certain, further investigation can be done by a human operator, and the data can be relabelled and used to improve the system.

As shown, the initial results are promising, but they should not be considered enough to draw definite conclusions on the reliability of this method. Further work is necessary in this area of research before a live system using this architecture could be used.

One question that needs to be answered is how the system deals with large amounts of data in a small amount of time. Large organisations are often the main targets of DDoS attacks, this is because the malicious actor generally wants to cause the most disruption possible. This means that the system will need to be able to deal with an extremely large amount of traffic, thus requiring incredibly fast packet processing. The prototype system designed in this investigation was trained and tested using packet flow data; this required an extra step in the processing of packets. Using packet flow data in a live system may not be reliable as by the time a set of packets has been collated into a flow, processed and then analysed, it may be too late and an attack may have already been successfully conducted. Further work is necessary here, perhaps using live network tests, to determine the efficacy of such a system. It may be that packet flow data is not a practical determiner for the recognition of attacks, and instead, individual packet data could be used.

Another area for investigation is how the system deals with different types of network topologies. The star topology was chosen in this investigation due to the simplicity of packet capture within it. All packets passed through a single point in the network, thus packet capture was straightforward, and a DDoS detection system running in this location would have access to all of this data. Furthermore, the network was remarkably simplistic, partially due to time constraints, but also due to this being only an initial investigation. It may be that a large organisation has a vast network with many gateways and multiple paths to a single server that could be targeted by a malicious actor. *Where would the detection system be placed in this case?* Perhaps the system can be copied into multiple locations, but there is a strong chance that a set of packets involved in an attack will not pass through just one of these locations. So, then we have the issue of the detection system only having access to a subset of the entire set of attack packet data – *would that be enough to recognise an attack?* It is a question that must be investigated before any live system with this architecture could be used.

Thought must also be placed into how the iterative learning process would function in reality. If the system encounters many unclassifiable samples, *how arduous would the process of manually investigating each of these be? Can it be done by a single person in a realistic amount of time, or would an entire team need to be on hand to deal with this?* The logic behind the process is sound, and would with some certainty lead to a robust detection system, but the practicalities must also be considered. Perhaps this issue could be dealt with by an external provider, whereby a business entrusts the improvement of their DDoS detection system to the external team. However, with this comes security issues, some organisations may not see the risk of sharing packet data with an external party as a reasonable tradeoff for improved DDoS protection.

In general, all of these issues to be investigated are problems of scale. Whilst they should still be investigated regardless, it could be suggested that this type of DDoS detection system may be used solely by smaller organisations to provide a solid method of preventing and mitigating attacks on their networks.

8 REFLECTIONS

The original aims of the project were to create a machine learning model to detect DDoS attacks using a pre-existing data set and to further test this model by using data from simulated DDoS attacks on the Cyber Range. These aims have been met and results show that the methodologies used appeared to have been successful.

During the project, I had the opportunity to learn a great deal about current methods of DDoS attack detection and prevention. Whilst much of the literature discussed methods of detection using machine learning, almost none used an architecture like that described and used in this project. Less still performed a simulated DDoS attack to collect data to test their model. The novelty of the project can be shown in this, and I believe that the architecture used in the detection system described in this project has a great potential to solve common issues encountered by traditional and even more recent methods of DDoS detection; such as the unknown attack type problem.

In addition, I believe I have learned a significant amount of useful information on machine learning, particularly regarding anomaly detection and unsupervised learning in general. I had implemented and used supervised machine learning models in previous projects, but prior to this project, I had no previous experience with implementing an unsupervised model and had no knowledge of anomaly detection. This project has allowed me to realise the potential utility of unsupervised learning, and how it can be combined with supervised learning to achieve goals that would not be possible with either method alone. The particular challenge involved here was to overcome that knowledge barrier and to learn how an unsupervised machine learning model can be used to identify patterns in data that are not explicit, a lot of reading was involved in this and it took a while for the idea to take hold. However, once the idea was clear in my mind, the suggested method of creating a hybrid system seemed like an excellent way of overcoming certain problems associated with single-method DDoS detection systems and I quickly got a clear idea of what needed to be done.

One aspect of the MSc Cyber Security & Technology course that particularly helped during this project was the knowledge I acquired on virtualisation, particularly the work done involving the Cyber Range. Prior to my study on this course, I had very limited knowledge of the core aspects of virtualisation and its potential uses. The work done for this project using virtual machines to create a virtual network that could be used to simulate activity and cyber attacks was a crucial part of the required novelty in this project, and it allowed me to test the system I had built with completely new data – meaning that the utility of the system in use on different networks could be demonstrated. However, due to time constraints and technical issues regarding access and use of the Cyber Range, the amount of work done in this regard was limited. It would have been interesting to test the system on multiple topologies, different target services and different kinds of attacks. This was unfortunate, but it provides scope for future work to be done.

The mentioned problems involving the Cyber Range were the biggest obstacle that had to be overcome in this project for me, and were unfortunately mostly out of my control. However, another major challenge in this project was the setting up of the virtualised network and its services. Having learned how to create a functional mail server, web server and DNS server for the project, I was provided with technical skills that will certainly be useful in my future career, and it gave me a further understanding of how these systems function in a deeper sense. This part of the project also allowed me to exercise my networking skills in a practical way that I had not had the opportunity to do previously.

I believe that an early error in my judgement was the expected amount of time required to collate and preprocess the data retrieved from the pre-existing data sets and the Cyber Range experiments for use in the training of the machine learning models and testing of the system. This was a challenging aspect due to the non-uniformity of the data, and the ambiguity of some of the labelling. I expected this part of the project to be relatively simple and found that it was actually very challenging, partly due to the processing capacity of the machine I used for this project, but also due to the need for a bespoke script to preprocess the data, and the opportunity for human error to creep in with this. I often found that results that I had believed to be accurate were not so on closer inspection due to a minor error made earlier in the process. This is an area I believe I would pay more attention to planning meticulously in future projects of this nature.

Furthermore, it was necessary to ensure that training and testing of the models were done in a way that conformed with standard scientific practices. This is something that I believe I managed to do, but not in an *ideal* manner. My understanding is that the most reliable way of testing a machine learning model is to use k-fold cross-validation testing. Indeed, this is what I had originally planned to do. However, due to the difficulties in preprocessing and the time spent in that regard, I felt I had to change my plans with this to meet the deadlines. I believe that the results I have are

still reliable, but this is an area that I would certainly look to improve, again by proper planning, in future work.

To conclude, I believe that this project has been a success and I have learned some useful technical and transferrable skills in its undertaking. Being able to spend time working with a powerful piece of technology in the Cyber Range was a great privilege and allowed me to further my understanding of virtualisation and networking in a way that would not have been possible otherwise. I also had the opportunity to perform a denial-of-service attack on a functional target, allowing me to experience the other side of cyber, gaining some insight into how such attacks can be performed and the related effects and signatures. Further, this project helped me improve my timekeeping, reading, reviewing and writing abilities, with the need to ensure that I met my target deadlines, attended meetings with the supervisor and had prepared notes to discuss, critically evaluated existing literature and wrote clearly and accurately to convey the rationale behind my decisions and the meaning of the results. These are all skills that will be useful in future projects I undertake, and in many other aspects of my career and life.

REFERENCES

Amazon (2022). Working at Amazon. [Online]. Available: https://www.aboutamazon.co.uk/news/working-at-amazon [Accessed: September 01, 2022].

Bajracharya, B. R. (2019). Detecting DDoS Attacks Using Logistic Regression. Master's thesis, Tribhuvan University, Kathmandu.

Balkanli, E., Alves, J., and Zincir-Heywood, A. N. (2014). Supervised Learning to Detect DDoS Attacks. In *Proceedings of 2014 IEEE Symposium on Computational Intelligence in Cyber Security (CICS)*, pages 1–8.

Bindra, N. and Sood, M. (2019). Detecting ddos attacks using machine learning techniques. *Automatic Control and Computer Sciences*, page 419–428.

Breunig, M.M. and Kriegel, H.-P. and Ng, R.T. and Sanger, J. (2000). LOF: identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*.

Canadian Institute for Cybersecurity, New Brunswick (2017a). CICFlowMeter (formerly ISCXFlowMeter). [Online]. Available: https://www.unb.ca/cic/research/applications .html#CICFlowMeter [Accessed: September 05, 2022].

Canadian Institute for Cybersecurity, New Brunswick (2017b). Intrusion Detection Evaluation Dataset (CIC-IDS2017). [Online]. Available: https://www.unb.ca/cic/datasets/ids-2017.html [Accessed September 20, 2022.

Canadian Institute for Cybersecurity, New Brunswick (2019). DDoS Evaluation Dataset (CICDDoS2019).

Chaykowski, K. (2015). Number of Facebook Business Pages Climbs to 50 million With New Messaging Tools. [Online]. Available: https://www.forbes.com/sites/kathleenchaykowski /2015/12/08/facebook-business-pages-climb-to-50-million-with-new-messaging-tools [Accessed: September 01, 2022].

Coppola, D. (2022). E-commerce revenue in the united kingdom from 2015 to 2021. [Online]. Available: https://www.statista.com/statistics/282162/e-commerce-annual-sales-in-the-united-kingdom-uk [Accessed: September 01, 2022].

DIATEAM (2022). DIATEAM. [Online]. Available: https://www.diateam.net/ [Accessed: September 12, 2022].

Dixon, S. (2022). Number of monthly active Facebook users worldwide as of 2nd quarter 2022. [Online]. Available: https://www.statista.com/statistics/264810/number-of-monthly-active-facebook-users-worldwide [Accessed September 01, 2022].

Ferguson, P. and Senie, D. (1998). RFC2267: Network Ingress Filtering: Defeating Denial of Service Attacks which Employ IP Source Address Spoofing. [Online]. Available: https://www.rfc-editor.org/rfc/rfc2267 [Accessed: September 02, 2022.

Fix, E. and Hodges, J. (1951). Discriminatory Analysis. Nonparametric Discrimination: Consistency Properties. *USAF School of Aviation Medicine*.

Gutnikov, A., Kupreev, O., and Shmelev, Y. (2022). DDoS attacks in Q1 2022. [Online]. Available: https://securelist.com/ddos-attacks-in-q1-2022/106358 [Accessed: September 01, 2022].

Hall, M. (1999). Correlation-based feature selection for machine learning. *University of Waikato*.

hMailServer (2022). hMailServer. [Online]. Available: https://www.hmailserver.com/ [Accessed: September 12, 2022].

Ho, T. (1995). Random Decision Forests. In *Proceedings of the 3rd International Conference on Document Analysis and Recognition. Montreal, QC.*

IBISWorld (2022). Percentage of Business Conducted Online. [Online]. Available: https://www.ibisworld.com/us/bed/percentage-of-business-conducted-online/88090 [Accessed: September 01, 2022].

Jin, C., Wang, H., and Shin, K. (2003). Hop-count filtering: an effective defense against spoofed DDoS traffic. In *Proceedings of the 10th ACM conference on Computer and communications security, Washington D. C*, pages 30–41.

Killalea, T. (2000). RFC3013: Recommended Internet Service Provider Security Services and Procedures. [Online]. Available: https://www.rfc-editor.org/rfc/rfc3013 [Accessed: September 02, 2022].

Li, Y. and Lu, Y. (2019). LSTM-BA: DDoS Detection Approach Combining LSTM and Bayes. In *Proceedings of Seventh International Conference on Advanced Cloud and Big Data (CBD)*, page 180–185.

Liu, F., Ting, K., and Zhou, Z.-H. (2008). Isolation Forest. In *Proceedings of Eighth IEEE Internation Conference on Data Mining*.

Mahjabin, T., Xiao, Y., Sun, G., and Jiang, W. (2017). A survey of distributed denial-of-service attack, prevention, and mitigation techniques. *International Journal of Distributed Sensor Networks*.

Microsoft (2021). What's New in DNS Server in Windows Server. [Online]. Available: https://docs.microsoft.com/en-us/windows-server /networking/dns/what-s-new-in-dns-server [Accessed: September 12, 2022].

NGINX (2022). What Is Load Balancing? [Online]. Available: https://www.nginx.com/resources/glossary/load-balancing [Accessed: September 02, 2022].

NumFOCUS (2022). Pandas. [Online]. Available: https://pandas.pydata.org [Accessed: September 03, 2022].

NumPy (2022). NumPy. [Online]. Available: https://numpy.org [Accessed September 03, 2022].

OffSec Services Limited (2022a). hping3 - Tool Documentation. [Online]. Available: https://www.kali.org/tools/hping3/ [Accessed: September 12, 2022].

OffSec Services Limited (2022b). Kali. [Online]. Available: https://www.kali.org/ [Accessed: September 12, 2022].

Park, K. and Lee, H. (2001). On the effectiveness of route-based packet filtering for distributed DoS attack prevention in power-law internets. *ACM SIGCOMM Computer Communication Review*, page 15–26.

Peng, T., Leckie, C., and Ramamohanarao, K. (2003). Protection from distributed denial of service attacks using history-based IP filtering. In *Proceedings of IEEE International Conference on Communications. Anchorage, AK.*, pages 482–486.

Python (2022). pickle - Python object serialization. [Online]. Available: https://docs.python.org/3/library/pickle.html [Accessed: September 11, 2022].

Roser, M., Ritchie, H., and Ortiz-Ospina, E. (2020). Internet. [Online]. Available: https://ourworldindata.org/internet. [Accessed: September 01, 2022].

Sahi, A., Lai, D., Li, Y., and Diykh, M. (2017). *An Efficient DDoS TCP Flood Attack Detection and Prevention System in a Cloud Environment*. PhD thesis, University of Southern Queensland, Toowoomba.

Schölkopf, B., Burges, C., and Vapnik, V. (1995). Extracting support data for a given task. Proceedings of First International Conference on Knowledge Discovery and Data Mining, Menlo Park, CA.

Schölkopf, B., Williamson, R., Smola, A., Shawe-Taylor, J., and Platt, J. (1999). Support Vector Machines for Novelty Detection. In *Advances in Neural Information Processing Systems 12, NIPS*, page 582–588.

scikit learn (2022a). Novelty and Outlier Detection. [Online]. Available: https://scikit-learn.org/stable/modules/outlier_detection.html [Accessed: September 03, 2022].

scikit learn (2022b). scikit-learn: Machine Learning in Python. [Online]. Available: https://scikit-learn.org/stable/.[Accessed: September 03, 2022].

scikit learn (2022c). sklearn.linear_model. LogisticRegression. [Online]. Available: https://scikit-learn.org/stable/modules/generated/ sklearn.linear_model/LogisticRegression.html [Accessed: September 10, 2022].

scikit learn (2022d). Supervised Learning. [Online]. Available: https://scikit-learn.org/stable/supervised_learning.html [Accessed: September 09, 2022].

Shawe-Taylor, J. and Cristianini, N. (2004). *Kernel Methods for Pattern Analysis*. Cambridge University Press, Cambridge.

Shieh, C.-S., Lin, W.-W., Nguyen, T.-T., Chen, C.-H., Horng, M.-F., and Miu, D. (2021). Detection of Unknown DDoS Attacks with Deep Learning and Gaussian Mixture Model. *Applied Sciences*.

The Apache Software Foundation (2022). Apache - HTTP Server Project. [Online]. Available: https://httpd.apache.org/ [Accessed: September 12, 2022].

Wireshark (2022). About Wireshark. [Online]. Available: https://www.wireshark.org/ [Accessed: September 12, 2022].