Security related self-protected networks: Autonomous Threat Detection and Response (ATDR)

by

Wessel Johannes Jacobus Havenga

A thesis submitted in partial fulfilment
of the requirements
for the degree of

Masters in Computer Science

in the

Department of Computer Science

at the

University of the Western Cape

October 2019

Supervisor: Professor Bigomokero Bagula

Abstract

Cybersecurity defense tools, techniques and methodologies are constantly faced with increasing challenges including the evolution of highly intelligent and powerful new-generation threats. The main challenges posed by these modern digital multi-vector attacks is their ability to adapt with machine learning. Research shows that many existing defense systems fail to provide adequate protection against these latest threats. Hence, there is an ever-growing need for self-learning technologies that can autonomously adjust according to the behaviour and patterns of the offensive actors and systems. The accuracy and effectiveness of existing methods are dependent on decision making and manual input by human experts. This dependence causes 1) administration overhead, 2) variable and potentially limited accuracy and 3) delayed response time.

In this thesis, Autonomous Threat Detection and Response (ATDR) is a proposed general method aimed at contributing toward security-related self-protected networks. Through a combination of unsupervised machine learning and deep learning, ATDR is designed as an intelligent and autonomous decision-making system that uses big data processing requirements and data frame pattern identification layers to learn sequences of patterns and derive real-time data formations. This system enhances threat detection and response capabilities, accuracy and speed. Research provided a solid foundation for the proposed method around the scope of existing methods and the unanimous problem statements and findings by other authors.

ATDR is a multi-layered approach that will meet the demands of the ever-changing threats landscape. At the lowest layer, network packet data is constantly learned to maintain the most optimal processing system during any change in load and conditions. With an optimal queue processing layer, the upper layers perform pattern recognition and identification. Collectively, the traffic data is optimally processed, while data patterns are identified and grouped into clusters, accurately and efficiently, for self-organized responses to threats.

A case study is used to demonstrate the effectiveness of the proposed ATDR approach through a series of tests using various kinds of data sets. The inspection and classification of network data into organized cluster groups enables granular automated management. The performance study shows that network traffic processing can be enhanced through round-robin scheduling with real-time optimal processing time slice adaptation. The security-related self-protected network can govern and control its own state of affairs for resilient cyber defense.

Security related self-protected networks:
ATDR: A Method for Automated Threat Detection and Response

Wessel Johannes Jacobus Havenga

Keywords

(Distributed) Denial of Service Attacks
DoS, DDoS and Malware Attacks
Traffic capture and packet analysis
DPI - Deep Packet Inspection
Queueing Theory
Machine learning
Neural Networking
Deep Learning
Pattern Recognition
Multi-vector attack detection
Anomaly detection
Network traffic identification and classification
DBSCAN
Cluster Analysis
Network optimisation
Self-organising networks
Artifical Intelligence (AI) Threat Hunting
Self-organised networks
Self-protected networks

Abbreviations

Table 1: Abbreviations

Abbr.	**Word/Phrase**
AI	Artificial Intelligence
ATDR	Autonomous Threat Detection and Response
APIC	Automated pattern identification and classification
CNN	Convolutional neural networks
CNNH	Convolutional neural network hashing
DBSCAN	Density-based spatial clustering of applications with noise
DMDBSCAN	One method of DBSCAN algorithm modification on Eps value to obtain optimal value automatically with different densities
DL	Deep learning
DN	Deep neural network
FIFO	First in first out
LRD	Long-range dependence
MAP	Mean average precision
PR	Pattern recognition

Symbols

Table 2: Mathematical Symbols Overview

Symbol	Description
e	Epsilon. Distance parameter that defines the radius to search for nearby neighbours
eps	A clustering parameter that specifies how close data points should be to each other to form part of the same cluster
k & K-means	K-means algorithm identifies k number of centroids, and then allocates every data point to the nearest cluster, while keeping the centroids as small as possible
minPTS	Minimum Data Points, signifies how many neighbours should be in close proximity to be considered a cluster

List of Figures

List of Tables

Contents

Chapter 1

INTRODUCTION

1.1 Introduction

With the growing ubiquity and complexity of data, cybersecurity has become a subject of utmost importance. Developing models capable of protecting the integrity of data as well as reducing cyber attacks is thus paramount. These threats/attacks, which can be introduced at different levels of the network, can be discovered by random or scheduled inspection of network systems. However, at high data rates, deep network packet inspection introduces latency, as packets have to queue to be processed. These delays are further exacerbated by payload encryption, which increases the processing time of packets and can also significantly affect inspection accuracy of legacy systems.

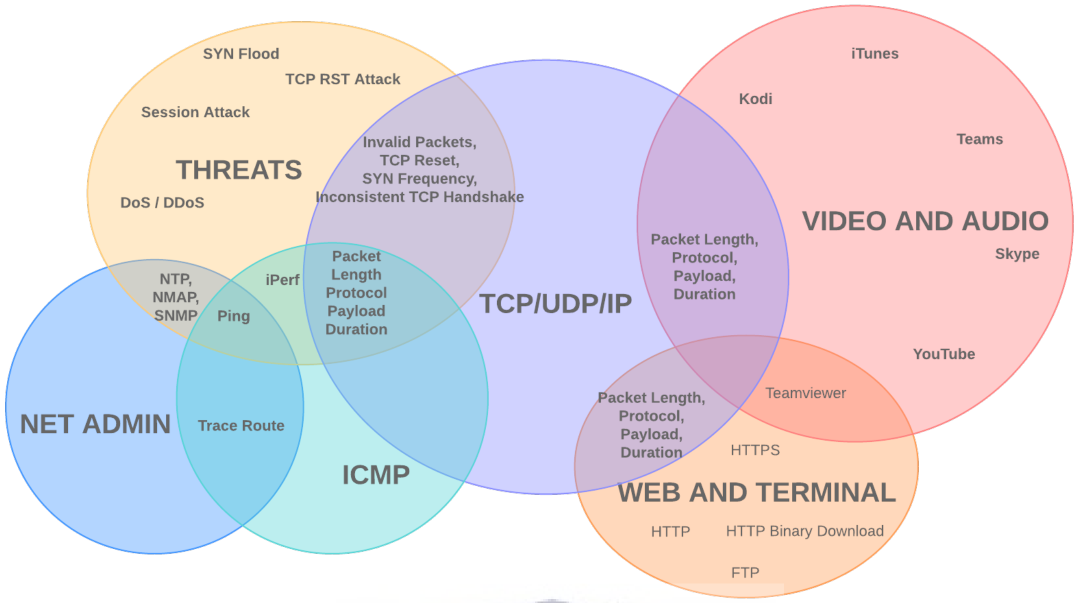


Figure 1.1: A Network Security Landscape

Figure 1.1 depicts the network security landscape, with potential threats to the different levels of a computer network. It shows attacks on prominent protocols such as Transmission Control Protocol (TCP), User Datagram Protocol (UDP) and Internet Protocol (IP), as well as the different tools that may be used by the network administrator to mitigate the impact of these attacks [2].

The recent increase in the adoption rate of high-speed connectivity solutions has resulted in a corresponding increase in network density and by extension the threat landscape. As a result, current traffic pattern recognition and classification systems require more time and continuous manual interventions from human experts to optimally calibrate requisite features and/or parameters. Undoubtedly, the efficacy of these methods is questionable, as human errors are inevitable. Automatic classifiers can be valuable in this regard, but they require "deep learning" of the network system, to accurately and efficiently recognise and classify threats. Achieving this goal is the crux of this research work, which is to automate the detection and response to anomalous and malicious traffic flow and feature management in real-time, with automatic and elastic scaling.

1.2 Research problem

Currently, embedding Machine Learning (ML) into network classification requires extensive manual and complex intervention from domain experts. Variance in capabilities of human experts limits the definition of best feature sets, identification of distinct patterns in data (annotation), as well as the design of accurate and optimal models that generalise well, especially in real-time scenarios. The effectiveness of the system to accurately identify patterns in real-time (as they are detected) is highly influenced by the limitation imposed by the human experts' intervention and calibration. Moreover, new protocols do not follow the rule of port registration, thereby resulting in increased error rates. Signature-based identification systems are efficient and accurate, but as protocol registration specifications change with time, the identification process must be changed accordingly and that becomes time consuming and labour-intensive.

1.3 Research aim and objectives

1.3.1 Primary aim

The primary aim of this research is to develop an autonomous threat detection and response (ATDR) system for computer networks, which is able to achieve high degrees of classification accuracy with minimal human intervention.

It seeks to ascertain whether ATDR can reduce manual expert input around classifiers for classification tasks, increasing the overall accuracy, efficiency and completeness of the classifiers required to detect threat vectors that include zero day threats in real-time, compared to existing methods.

1.3.2 Research objectives

According to [3], the success of IP traffic classification can be measured using two metrics: *completeness* and *accuracy*. *Accuracy* measures the number of correct matches a signature makes and the confidence level thereof, while *completeness* is the availability of a signature (classifier) for each threat protocol present in the network.

With the research aim stated above and the criteria for IP classifiers spelt out, the objectives of this work are thus:

- To determine if ATDR can accurately and automatically detect new threat patterns in dynamic, uncategorised noisy and large data sets.
- To determine whether ATDR can be implemented as a general method, supporting integration and application across a wide range of problems compared to other ML problems.
- To determine whether ATDR can automatically derive customised classifiers, with accuracy results that are comparable with (or improvement upon) other ML methods implemented by human experts for similar tasks.
- To use case studies to determine the accuracy and effectiveness of ATDR with regards to IP traffic classification.

1.4 Contributions

The contributions of this thesis are in three fold: 1) to propose a method for ATDR for self-protected networks using a combination of unsupervised machine learning and Deep learning models, 2) leverage on big data analysis, pattern recognition and deep packet inspection to design ATDR as an intelligent and autonomous decision-making system, and 3) derive a system capable of accurate and fast real-time data threat detection and response, with minimal human intervention.

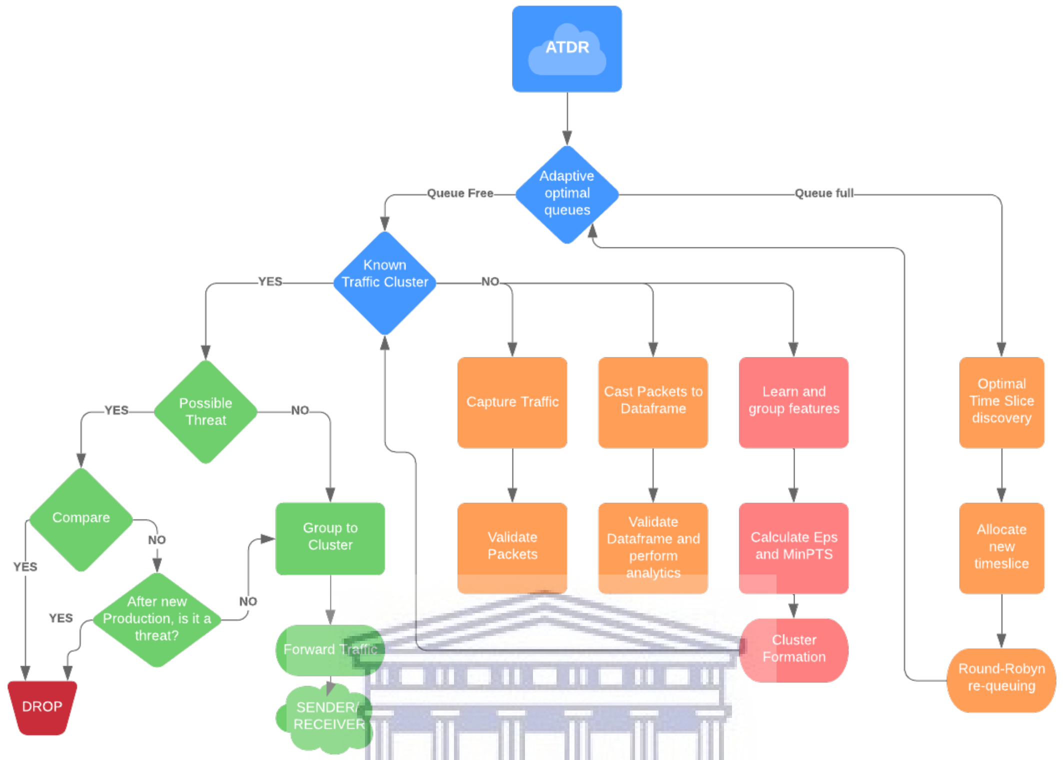


Figure 1.2: High-Level Overview of the ATDR Model

The proposed ATDR model is aimed at providing an optimal queuing deep packet inspection model for traffic identification and classification that will contribute toward overcoming current challenges, as stated in [2].

The model consists of different functional workflows as can be seen in Figure 1.2. Firstly, the network packets processing is aimed to be optimally tuned and dynamically changeable according to workload and traffic conditions. With optimal processing in place, deep learning can take place at the other layers to recognise traffic patterns, which would be used to accurately and efficiently classify traffic for optimal network traffic steering. The final layer inspects the traffic for anomalous and malicious behavioural patterns and then enforces mitigation and prevention of further processing of unwanted anomalous packets for resource preservation and optimisation [4].

The network packet scheduling is based on Markov Chains, because it is designed for when the future of the process is independent of the past given only its present value.

1.5 Thesis outline

Chapter 1 of this thesis explains the concept of ATDR, itemising the aim and objectives of the research work and importantly the intended contribution of this study to the science community and the world at large. Chapter 2 describes Machine Learning techniques as cutting-edge approaches in a broad range of data classification tasks. It also shows that the majority of these techniques are strongly dependent on the intervention of human experts for feature set extraction, optimisation, data set annotation, and model design processes. These inefficiencies result in less reliable and incomplete systems, where overly complex classifiers predisposed to human error are highly likely. Chapter 3 outlines a novel method for addressing these deficiencies with the design and implementation of ATDR, wherein optimised feature selection, pattern discovery, and optimised classifier production processes are performed automatically. In Chapter 4, the efficacy of this new method is tested using different case studies, including: IP Traffic Classification and anomaly detection. Finally, Chapter 5 provides the conclusion of this research and related future work that aims to further enhance the contribution of the research.

Chapter 2

Literature review

2.1 Literature overview

Machine learning (ML) is a sub-field of Artificial Intelligence that is concerned with learning from data or experience and has gained widespread adoption over the past decade. This increased interest and adoption can be attributed to two main factors: the ubiquity of data and the availability of adequate computational power for data analysis [3]. It is important to note that the computational power per dollar has increased exponentially at an average rate of about 55 percent per annum, since 1940. This increment created an avenue for ML model such as Artificial Neural Networks (ANN), which easily adapt to complexity situations to be explored.

Deep Learning (DL) networks are more complex forms of ANN and consist of multiple hidden layers, providing per-level non-linearity at each of the layers. Other general classifiers are adapted from the complexity of these multilayered models to avoid conventional ANN pitfalls such as over-fitting. Multiple layers enable the algorithm to more accurately model high-level abstractions found in complex, large data sets. Classification tasks are complimented by ML due to the ability to model high-dimensional, complex data sets. A prevalent constraint of applying the technology in each task is it's dependency on human experts for successful implementation. Greater deployments of ML-based classification techniques, that cover a broader range of tasks, over multiple domains, will be achieved if these obstacles can be overcome and it is imperative to ensure that the algorithms behave predictably. Discussed below are the different models/techniques that have been used by various researchers in the analysing of large data with varying levels of complexity.

2.2 Hashing model

Hashing models generally transform character(s), string(s), key words or streams of data into another value, for security purposes to ensure credibility and/or integrity.

2.2.1 Deep hashing learning networks

Deep hashing learning networks use Convolutional Neural Networks (CNN) to learn hashing codes [5]. CNNs have been extensively utilised for diverse visual recognition and exceed human-level performance in many tasks, including recognising traffic signs, faces and hand-written digits. In [5] the authors encoded images into multiple levels of representation using deep CNN. This enabled the discovery of complex structures hidden behind high-dimensional data. For classification tasks, the important aspects of the input are reserved by the higher layers of representation for discrimination and irrelevant variations are inhibited.

This study leverages on the learning capability of deep CNNs to automate threat-defining features in computational networks.

2.2.1.1 Hashing model configuration

For a given data set $(x_1, x_2, ..., x_n)$ with $x_i \epsilon R^d$, let $\{y_i\}_i^n = 1$ with $y_i \epsilon \{0,1\}^m$ be the binary code for each input. In general [5] assumed different bits are independent of each other, that is $y_i = [h_1(x_i), h_2(x_i), ..., h_m(x_i)]^T$ with m independent binary hashing functions $\{h_k(.)\}_k^m = 1$. Each bit has a 50% chance of being either one or zero. The authors then sought to minimise the average Hamming distance between similar pairs.

2.2.1.2 Result analysis

The proposed methods are based on the open-source Caffe [37] framework. Filters of *32, 32, 64* were used with *size 5 * 5* in the first, second and third convolutional layers with each using a ReLU activation function. The hash mapping layer is located at the top of the third pooling layer, followed by a compression sigmoid layer. Results in Table 2.1 and Figure 2.1 reveal the precision-recall curves for KSH and BRE obtained by [5]. They also compared their result with CNNH, with theirs showing a 0.2% gain over CNNH when using the MNIST data set and an 18% to 27%

gain on CFIR-10. This shows that CNN-based methods can achieve better results compared to conventional methods.

Table 2.1: mAP on MNIST and CIFAR-10 data set, w.r.t different number of bits

method	MNIST(MAP)				CIFAR-10(MAP)			
	16bits	24bits	32bits	48bits	16bits	24bits	32bits	48bits
LSH	0.250	0.284	0.310	0.430	0.298	0.344	0.331	0.389
SH	0.347	0.383	0.393	0.387	0.352	0.355	0.379	0.381
PCAH	0.351	0.344	0.332	0.309	0.291	0.280	0.272	0.261
PCA-ITQ	0.515	0.550	0.581	0.610	0.427	0.445	0.453	0.469
SKLSH	0.182	0.231	0.218	0.256	0.288	0.312	0.334	0.394
KSH	-	0.891	0.897	0.900	0.303	0.337	0.346	0.356
BRE	-	0.593	0.613	0.634	0.159	0.181	0.193	0.196
CNN+	-	0.975	0.971	0.975	0.465	0.521	0.521	0.532
Ours	**0.992**	**0.993**	**0.995**	**0.993**	**0.714**	**0.718**	**0.736**	**0.728**

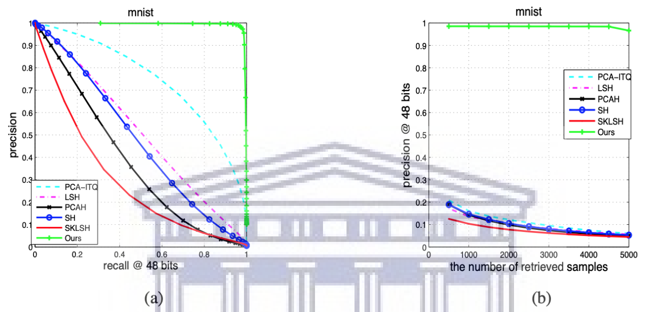


Figure 2.1: Quantitative comparison results on CIFAR-10. (a) Precision-recall curves with 48 bits, (b) Precision-recall curves w.r.t numbers of top returned images

2.2.1.3 One hot encoding model overview

According to [5], a good code for hashing is satisfied by 3 conditions, namely

1) projecting similar pairs in data space to similar binary code words in Hamming space

2) a small number of bits to encode each sample

3) little computation for input projection.

2.2.1.4 One hot encoding

One hot encoding is a method to encode integer features using one-hot also known as one-of-K scheme. One hot encoding is a process whereby categorical variables are converted into a form that could be provided to ML algorithms to improve prediction and accuracy. The numerical value of the entry in the data set is represented by a categorical value. Other algorithms assign values to categories from *0* to *N-1* categories. This is not sufficient as the higher categorical value is assumed to be the better category and could cause prediction errors. A one hot encoder can be used as a tool to mitigate this type of error. It converts the category to binary and includes it as a feature to train the model.

The hashing models/techniques had no limits around data scale and could generate hash codes with little computation. This can further be boosted by Graphic Processing Unit (GPU) acceleration and multi-threading, which this research contributes with an optimal processing engine using the First-In-First-Out (FIFO) queue processing that can scale into multiple processing threads as described in Chapter 3.

2.3 Machine learning

Machine learning (ML) is a process whereby machines are able to "intelligently" perform tasks that require human intuition, by observing data to discover hidden patterns. ML is a collection of powerful techniques for data mining and knowledge discovery in data (structured or unstructured). A learning machine has the ability to learn automatically from experience, refine and improve its knowledge base.

ML can broadly be categorised into: Classification (or supervised learning), Clustering (or unsupervised learning), Association, and Prediction. [6, 7]

2.3.1 Data mining

Finding data patterns is common-place; however, capacities of modern databases and the overall overwhelming amount of data stored within them has made it cumbersome to process and analyse these data. Furthermore, the larger the database size the lower the ability of humans to manually process and make sense of it.

The author in [8] defines data mining as the process of automatically or semi-automatically discovering patterns in data. Structural patterns capture the decision structure in an explicit way and help to describe the data. The authors term this process "Decision Trees". With "decision trees", a sequence of decisions along with the resulting recommendation as a popular means of expression is made. They also introduced "decision tree induction" and "nearest-neighbour" methods, noting that most learning algorithms use statistical tests when constructing trees or rules, and Machine learning techniques and models are validated by statistical tests.

Alternatives to "Decision Trees" are "covering approaches". Covering approaches lead to a set of rules rather than to a decision tree and at each stage you identify a rule that "covers" some of the instances. The study in [8] describes two interesting Fielded Application Machine learning techniques namely "Decisions including judgement" and "load forecasting". Decisions including judgement statistical methods are used to clearly distinguish "accept" and "reject" cases, while Load Forecasting is a prediction of future demands estimated on historical data.

Another approach is the use of expert systems learning approaches, which use induction algorithms to find rules automatically that human experts would have derived with their knowledge. [8] illustrated analogies where data mining can be used. An example was the production of women's embryos by fertilising the ovaries with sperm, and noting the 60 character traits that exist pertaining to the quality of ovaries' eggs. Another illustration is the collection of breeding and milk production history of dairy cows considering the herd's differentiating characteristics such as age, behaviour and production success. They noted that Machine learning ascertains what factors form successful outcomes and results for future comparisons.

2.3.2 Internet traffic classification using ML

The authors in [7] suggested that current research should focus on IP traffic classification techniques that do not rely on well-known TCP or UDP ports. Traffic at the network layer has statistical properties (such as the distribution of flow duration, flow idle time, packet inter arrival time and packet lengths). These properties are unique for certain classes of applications and enable different source applications to be distinguished from each other. The need to deal with traffic patterns, large data sets and multi dimensional spaces of flow and packet attributes is one of the reasons for the introduction of ML techniques.

In 1994, ML was first utilised for Internet flow classification in the context of intrusion detection. ML is the process of finding and describing structural patterns in a supplied data set. ML takes input in the form of a data set of instances (also known as examples). In the networking field, consecutive packets from the same flow might form an instance, while the set of features might include median inter-packet arrival times or standard deviation of packet lengths over a number of consecutive packets in a flow. A class usually indicates the application or group of applications the IP traffic belongs to. Instances are usually multiple packets belonging to the same flow. Features are typically numerical attributes calculated over multiple packets belonging to individual flows.

Feature selection algorithms can be broadly classified into filter method or wrapper method. Filter method algorithms make independent assessment based on general characteristics of the data. Wrapper method algorithms, on the other hand, evaluate the performance of different subsets using an ML algorithm that will ultimately be employed for learning. The results are therefore biased toward the ML algorithm used. The 'flow statistics processing' step involves calculating the statistical properties of these flows (such as mean packet inter- "arrival time, median packet length and/or flow duration) as a prelude to generating features. The authors, [7], mention that when evaluating unsupervised ML schemes in an operational context, it is relevant to consider how clusters will be labelled (mapped to specific applications), how labels will be updated as new applications are detected, and the optimal number of clusters (balancing accuracy, cost of labelling and label lookup, and computational complexity). Important traffic classification should occur as the traffic is flowing or within a short period of time. Numerous applications (such as multiplayer online games or streaming media) exhibit different (asymmetric) statistical properties in the client-to-server and server-to-client directions. An inefficient classifier may be inappropriate for operational use regardless of how quickly it can be trained and how accurately it identifies flows. A model may be considered portable if it can be used in a variety of network locations, and robust if it provides consistent accuracy in the face of network layer perturbations such as packet loss, traffic shaping, packet fragmentation, and jitter. Also, a classifier is robust if it can efficiently identify the emergence of new traffic applications.

Ref. [9] in 2005 proposed the AutoClass, which is an unsupervised Bayesian classifier, using the EM algorithm to determine the best clusters from a data set. AutoClass can be preconfigured with the number of classes (if known) or it can try to estimate the number of classes itself. Firstly, packets are classified into bidirectional flows and flow characteristics are computed using NetMate. A metric called intra-class homogeneity, H, for assessing the quality of the resulting classes and classification is introduced. H of a class is defined as the largest fraction of flows on one application in the class. The overall H of a set of classes is the mean of the class homogeneities. The goal is to maximise H to achieve a good separation between different applications. The authors used accuracy (Recall) as an evaluation metric. Median accuracy is larger than or equal to 80% for all applications across all traces. TCP-based application identification is performed using Simple K-Means.

Ref. [10] proposed a technique using an unsupervised ML (Simple K-Means) algorithm that classified different types of TCP-based applications using the first few packets of the traffic flow. The rule worked as follows: the Euclidean distance between the new flow and the centre of each predefined cluster is computed, and the new flow belongs to the cluster for which the distance is a minimum. The flow duration in the missing direction is estimated as the duration calculated with the first and the last packet seen in the observed direction. The number of bytes transmitted is estimated according to information contained in ACKs packets. The number of packets sent is estimated with the tracking of the last sequence number and acknowledgement number seen in the flow, with regard to the MSS. Results show that the algorithm produces the best overall accuracy when compared to K-Means, DBSCAN and AutoClass, while DBSCAN had the highest precision value for P2P, POP3 and SMTP (lower than AutoClass for HTTP traffic) [1].

Comparing AutoClass with Naive Bayes on nine application classes (HTTP, SMTP, DNS, SOCKS, IRC, FTP control, FTP data, POP3 and LIMEWIRE), AutoClass resulted in an average overall accuracy of 91.2% while the Naive Bayes classifier has an overall accuracy of 82.5%. AutoClass also performs better in terms of precision and recall for individual traffic classes.

With respect to ranking, From literature, algorithms can be ranked in descending order of classification speeds as: C4.5, NBD, Bayesian Network, Naive Bayes, NBK. When ranked in descending order in terms of model build time: Naive Bayes, C4.5, Bayesian Network, NBD, NBK. Naive Bayes, AdaBoost and Maximum Entropy should be looked at for application signature building algorithms. Results show that BLINC can classify 80% to 90% of traffic flows with more than 95% flow accuracy. The first test, based on Pearsons Chi Square test, detects Skypes fingerprint through analysis of the message content randomness introduced by the encryption process. The second test, based on the Naive Bayes theorem, detects Skype traffic based on message size and arrival rate characteristics. There are still open questions as to how well they can maintain their performance in the presence of packet loss, latency jitter, and packet fragmentation. Therefore, the use of a combination of classification models is worth investigating.

2.3.3 Use of multiple sub-flows for ML classification of IP networks

The authors in [6] infer that current ML-based IP traffic classification algorithms consider either the first few packets or full flows; however, real-world scenarios require a classification decision well before the flow has finished. This implies that classification must be performed by using statistics derived from the most recent N packet taken at any point in a flow's lifetime. The authors further proposed to use a set of sub-flows, with windows as small as 25 packets long, extracted from full flow examples and then optimising with Naive Bayes ML algorithm that will result in excellent performance.

[6] suggests future use of unsupervised ML algorithms to identify optimal sub-flows for training. Traffic classification can be a core part of automated intrusion detection systems used to detect patterns of denial of service attacks, trigger automated re-allocation of network resources for priority customers, or identify customer use of network resources that in some way contravenes the operator's terms of service. Governments are lately also defining ISP obligations with respect to 'lawful interception' (LI) of IP data traffic. Customers may use encryption to obfuscate packet contents including TCP/UDP port numbers. Governments may impose privacy regulations constraining the ability of third parties to lawfully inspect payloads at all. Deep inspection of every packet will require commercial devices with repeated updates. Newer approaches classify traffic by recognising statistical patterns of observable patterns, such as inter-packet arrival times and typical packet lengths. Their ultimate goal is either to cluster IP traffic flows into groups that have similar traffic patterns or classify one or more applications of interest. Classification involves two stages: training the ML algorithm to associate sets of features with known traffic classes (creating rules) and applying the ML algorithm to classify unknown traffic using previously learned rules. This study focuses on the practical application of ML algorithms to traffic classifiers deployed in operational IP networks.

ML classification algorithms all assume that a 'class' of traffic can be identified using statistical analysis of traffic features. Unsupervised ML algorithms allocate flows to classes based on clustering of similar feature values. Supervised ML algorithms use examples of IP traffic, matching the class of traffic that is later to be identified in the network. Reducing the time taken to detect traffic of interest implies reducing the number of packets that must pass the monitoring point before classification can be achieved. Also, re-calculating features over a sliding window of N packets requires buffering of the most recent N packets. Minimising the number of buffered packets per flow provides a beneficial reduction of physical memory requirements. Naive Bayes is a well-understood supervised learning algorithm whose classification approach is based on probabilistic knowledge. Traffic flows are bidirectional streams of packets between a given pair of hosts. A flow is defined using the 5-tuple of source and destination IP addresses, protocol (TCP & UDP) and the source and destination ports. For UDP traffic a flow is considered to have stopped when no more packets are seen for 60 seconds. For TCP traffic, a flow is stopped when the connection is explicitly turned down or no packets are seen for 60 seconds. Online gaming traffic seen at a server can exhibit three different phases: clients probing the server, clients connecting to the server and clients actually playing a game on the server. Precision held steady at 98% when trained on the multiple sub-flows (similar to the single sub-flow model).

ML has also been used with the sliding window technique, which is a sequential learning methodology that converts a sequential learning problem into a classical learning problem. A window classifier hw maps an input window of width w into an individual output value y. The window classifier hw is trained by converting each sequential training example (x_i, y_i) into windows and then applying a standard ML algorithm. A new sequence x is classified by converting it to windows, applying hw to predict each y_t and then concatenating the Y_t to form the predicted sequence y. The obvious advantage of this sliding window method is that it permits any classical supervised learning algorithm to be applied. While the sliding window method gives adequate performance in many applications, it does not take advantage of correlations between nearby y_t values. Specifically, the only relationships between nearby y_t values that are captured are those that are predictable from nearby x_t values. If there are correlations among the y_t values that are independent of the x_t values, then these are not captured.

2.3.4 Classification using Bayesian analysis and neural networks

Authors in [11] introduced Internet traffic identification as an important enabler for 1) future traffic matrices and demand management, 2) malicious traffic behaviour recognition and 3) the development of more realistic traffic models. They developed a traffic classifier that can achieve high accuracy across a range of application types without any destination, source, host or port information by using supervised ML based on a Bayesian trained neural network. The training data categories were extracted from packet headers with training and testing done using features derived from packet streams. Their proposed technique offers wider application as it does not inspect the packet content, but rather only commonly available packet header information is inspected. The authors argue that application classification schemes are inaccurate due to only limited packet header information being available. Using supervised ML to train a classifier, the predicted category was compared to the actual category of each object. The authors claim that their results presented much more accuracy than that of port-based mechanisms. The valuable contributions from [11] include the use of a Bayesian framework with a neural network model that allows identification of traffic with 99% accuracy without monitoring port, IP and host information. The authors further stated that network operators and users have a broad interest in observing network anomalies to reveal malicious traffic. Their process of combining host knowledge and content verification through contact with the users, allowed the original data to be classified, manually, with very high accuracy.

Furthermore, site-specific information such as the role of machines, the access to the content of packets and access to the system administrator and user community may not be assumed. A number of properties are referred to and used such as client and server port of each flow, along with behavioural characteristics. The aforementioned properties enabled the authors to differentiate between the different traffic classes. Network training consists of choosing the best weights of the network for a particular problem. Entropy is where the network is different to the prediction it is making. The authors confirm satisfaction when they note that when the network makes a false classification, low confidence is indicated. In a classification situation in which false predictions are to be minimised, the entropy of the distribution can be used to reject a prediction. The percentage of prediction rejections (based on entropy) was then used to verify the behaviour, and the accuracy was recorded. A maximum value of the distribution of the classes could be set but given a high entropy, the predictions are discarded and it is inferred that the classifier is unable to predict the class accurately for that flow. Features that are derived from packet headers, when treated as interdependent, can provide an effective method for the identification of network-based applications [2]. They conclude with results that proved that it is possible to reverse the data to discover the application in use. The authors also note that classification accuracy declines over time as the composition of Internet traffic changes.

2.3.5 Traffic classification using clustering

The authors in [10] highlight the necessity for early identification of TCP flows without using port numbers. They further mention that the current modern techniques cannot identify traffic without looking at the full TCP flows. They then propose a method that recognises traffic after monitoring only 5 packets. The size of the first few packets is a good predictor of the application associated with a flow because it captures the application's negotiation phase. A single application can have multiple behaviours that should each have a model. FTP is an example of a multi-modal application. Encryption is however a challenge, as inspecting packet payload can only be accurate if the packets are not encrypted.

In applying clustering, [10] explains that representation flows are represented by points in a P-dimensional space where each packet is associated to a dimension. The coordinate on the dimension p is the size of packet p in the flow. A heuristic is used and the Euclidean distance between the new flow and the centre of each predefined cluster is computed. The cluster with the minimum distance is chosen. The processing of training flows with a payload analysis tool is able to accurately determine the application associated with each flow. Their classifier takes the series of packet headers for both directions of an edge link as input. A packet analyser extracts the 5-tuple (protocol, source IP, destination IP, source port, destination port) and the packet size. The analyser filters out control traffic (the three packets of the TCP handshake) and stores the size of every packet in both directions of the connection. With the size for the first P packets of the connection ascertained, it sends this information to the flow conversion module, which maps the new flow to a spatial representation. The cluster assignment module searches all the cluster descriptions to find the best fit for this new flow and the application identification module selects which application is most likely associated with the flow given the set of applications that compose the cluster. A prototype classifier was built with Matlab.

2.3.6 A survey on Internet traffic identification

Internet traffic measurement has improved enormously over the years. This is due to increased network access speeds and growth in connected (data dependent) applications. These changes affect the work of network administrators and service providers with respect to resource demands. Traffic analysis is separated into packet-based and flow-based categories. Techniques that will be revised include signature-matching, sampling and inference. Characterisations of Internet traffic provide insight for various network management activities, such as capacity planning and provisioning, traffic engineering, fault diagnosis, application performance, anomaly detection and pricing [12].

The availability of broadband user connections is continually growing and has opened up new ways of resources usage for both home and small businesses, including an increase in a wider range of services being consumed like voice over IP, e-commerce, Internet banking, etc. As far as broadband residential users and access providers are concerned, measuring traffic to and from the customer base is essential for understanding user behaviour. Measurement strategies are key for detecting anomalous traffic, design and validation of new traffic models and optimal capacity planning. To differentiate, network measurement is about data gathering and counting, while application identification is the recognition and classification of some traffic characteristics. Traffic measurements can be divided into active and passive measurements or online and offline strategies. With online, traffic analysis is performed while capturing the traffic, while with offline, data traces are stored and analysed later.

Flow-based measurement deals with a compilation of unidirectional streams of packets passing through a given router. There are three possible ways to perform packet capture, namely 1) cable splitter to capture without affecting the traffic, 2) port mirroring to clone the port for monitoring and 3) active equipment can be installed inline. Sampling techniques are crucial for scalable Internet monitoring due to large volumes of high-speed link capacity data. Sampling techniques may be divided into systematic, random and stratified sampling. The simplest sampling process is uniform 1/N sampling (systematic sampling), where the first one of every $Npacketsisretained.Anotherverysimpleformisrandom1/Nsampling(randomsampling),wherearandompacketofever$ packets is retained. Stratified sampling technique is based on the idea of increasing the estimation accuracy by using some prior information: it performs an intelligent grouping of the elements of the parent population and many levels of grouping may be considered. Heavy hitters (large flows) stay as heavy hitters for most of their lifetime and mice (small flows) rarely become heavy hitters as revealed in the elephants and mice phenomenon study by duration, longer flows are called tortoise and shorter ones are called dragonfles. By rate, heavy flows are called cheetahs and light ones snails and by business, bursty flows are called porcupines and non-bursty ones stingrays; but these are also called alpha and beta traffic.

The Internet community is aware that a small percentage of flows are responsible for a high percentage of the traffic volume. This paper analyses the relationships between the different heavy hitters and concludes that there is a strong (over 80%) correlation between flow rate and bursting, between flow size and rate and between flow size and bursting, while there is a small correlation between flow duration and all the other metrics. It is well known that aggregate traffic exhibits long-range dependence (LRD) correlations and non-Gaussian marginal distributions. LRD traffic signals that the traffic exhibits only small variations on the intensity of self-similarity over different timescales. Stripping the alpha (bursty spikes) traffic from an aggregate trace leaves a beta traffic residual that is Gaussian, LRD, and shares the same fractal scaling exponent as the aggregate traffic. Ignoring flash flows will eventually improve the performance and accuracy of flow/volume-based prediction systems.

2.3.7 Application behaviour characterisation for fast and accurate large-scale traffic classification

System security and network management have critical dependency on Network Traffic Classification according to [13]. Furthermore, with modern applications changing to peer-to-peer (P2P) based protocols with encryption, the accuracy of current traffic classification techniques becomes less effective. [13] proposed an Application Behaviour Characterisation (ABC) System, extracting application behaviour using an effective classification algorithm that combines multiple flows of the same application type in a hierarchical manner. This system will provide good efficiency and high performance. New distributed applications with collaborative protocols among multi-parties (e.g., proprietary P2P, or peer-server hybrid systems) make existing techniques less effective because of the encryption and dynamic random port number usage and the authors reckon this is still an issue. Payload-based techniques that classify traffic by comparing packet payloads with known signatures are used to mitigate the random port usage issue that affects current classification methods. Two methods are used to match payload to signatures, namely: 1) string-based matching and 2) regular-expression matching. String-based matching supports many protocols including P2P protocols and HTTP, and can use fast multi-pattern matching algorithms.

2.4 Deep learning in network traffic identification

Most IP network systems are based on features that include port numbers, static signatures and characteristics. The challenge is to find the features in the traffic flow but the process is time-consuming and current methods do not cater for unknown protocols. The authors in [2] proposed a method using deep learning and neural networking and the results reflect that the approach works well for protocol identification, feature learning and anomalous protocol detection. The application of their method includes feature learning, protocol classification, anomalous protocol detection and unknown protocol identification. Earliest methods used special or predefined ports, with HTTP port 80 and HTTPS port 443 as an example. Recent approaches are based on automatic classification based on statistical features and ML. These approaches depict features around traffic transmission, such as time interval between packets, packet size, repeating pattern and so forth.

The features are then parsed through classifiers, such as Naive Bayes, Decision Tree and Neural Networks; with the objective of detecting the relevant features.

An Artificial Neural Networks (ANN) is as a ML inspired by the biological neural network in the brain and has been widely used for pattern recognition. Deep learning is a form of advanced ANN and includes methods such as Deep Neural Networks (DNN), Convolutional Neural Networks (CNN), Stacked Auto-Encoders (SAE) and Deep Belief Networks (DBN). Deep learning consists of efficient algorithms for semi-supervised or unsupervised feature learning and hierarchical feature extraction.

The number of layers is always three and the nodes in the output layer are the same as the in input layer. The middle layer consists of new features that have lower-dimensional representations, meaning that data can be reconstructed after complicated computations. Deep networks can be made up by stacking these structures.

In each training, results of the middle layer are cascaded, forming a new structure called a Stacked Auto-Encoder (SAE). In cyber security, a popular approach is defining features and signatures by experts' experience. The main difference between SAE and ANN is in the type of data set used. In ANN models, labels are necessary whereas in SAE they are not; they are in essence supervised and unsupervised respectively.

Some critical steps in ANN and DL are briefly discussed below.

2.4.1 Feature learning

This encompasses feature extraction and feature selection, both of which are necessary for both ANN and DL.

2.4.2 Feature extraction

With ANN models, the information in the hidden layers is automatically chosen and merged by the nodes in the outer layers. This process is known as raining or parameter learning. With the exception of the output layer, all nodes in any layer can be used as features. Nodes in the input layer are known as the original features.

2.4.3 Feature selection

Optimal parameters are achieved when the error cost is very low (about 10^{-3}) and stable at 50 to 60 epochs. A weight parameter W represents the contribution of the original parameter x to the hidden layer feature h. The magnitude of parameters, contribution is the importance of every byte in x. The sum of all absolute weights for every node in the input layer selected is then fed into the activation function for that layer.

2.5 Literature review conclusion

Following research of various domains in data learning and classification, it can be seen that researchers address the vast increase in density of network traffic differently. Identifying flow data, especially in real time, is an important problem with port-based, signature-based and statistical-feature-based identification being the mainstream approaches. However, they are either inaccurate or rely on human expertise. Signature-based is a portion of payload data that is static or distinguishable for applications which can be described as sequence of strings with a theoretical error rate lower than 10%. Signature-based is efficient with a high accuracy, but when protocol registration and specifications change over time, signature validation must start over and it then becomes time-consuming and labour-intensive.

Complementary to the goal of this research, supervised ANN methods are found to have two primary applicable advantages, which are: 1) autonomy, which reduces the need for manual human intervention, such that once input of the model and stopping criterion of the iteration are determined, the model will train automatically, and 2) dimensionality reduction, which ensures that once training is completed, relevant features get mapped to a new space and redundant information is filtered out.

Chapter 3

ATDR - DESIGN AND IMPLEMENTATION

3.1 ATDR optimal processing architecture

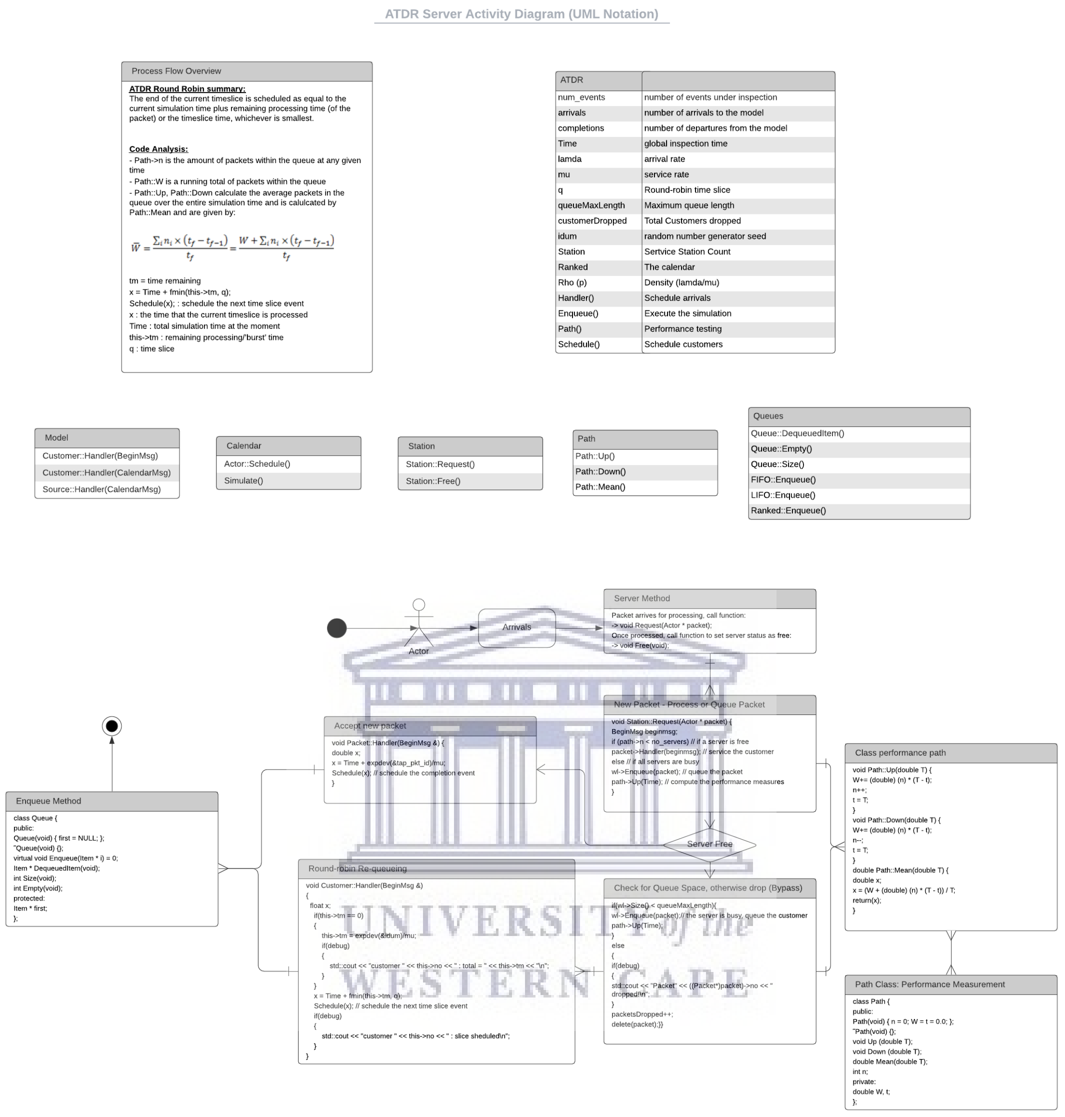
ATDR architecture is primarily based on two fundamental concepts: 1) queuing theory, and 2) round-robin scheduling.

3.1.1 Optimal traffic processing with M/M/1/N queuing theory

Queuing theory was initially created by Agner Krarup Erlang to describe the Copenhagen telephone exchange with the mathematical study of waiting lines [14] but has been adapted for application in various domains. Queue processing plays a fundamental role in the performance and stability of many processing systems used in various fields [15]. To avoid delay on the overall infrastructure and negative impact on user experience, the rate at which demands arrive needs to be timed and efficiently matched with the rate at which these requests are processed by the server's processor and for this we will consider the M/M/1/N system which represents Poisson.

3.1.1.1 Parameters

For the system processing model of the ATDR, we will consider a single server with multiple traffic queues. Figure 3.1 gives a high level overview of the ATDR process, with components described as follows:



Process Flow Overview

ATDR Round Robin summary:
The end of the current timeslice is scheduled as equal to the current simulation time plus remaining processing time (of the packet) or the timeslice time, whichever is smallest.

Code Analysis:
- Path->n is the amount of packets within the queue at any given time
- Path::W is a running total of packets within the queue
- Path::Up, Path::Down calculate the average packets in the queue over the entire simulation time and is calulcated by Path::Mean and are given by:

$$\bar{W} = \frac{\Sigma_i n_i \times (t_f - t_{f-1})}{t_f} = \frac{W + \Sigma_i n_i \times (t_f - t_{f-1})}{t_f}$$

tm = time remaining
x = Time + fmin(this->tm, q);
Schedule(x); : schedule the next time slice event
x : the time that the current timeslice is processed
Time : total simulation time at the moment
this->tm : remaining processing/'burst' time
q : time slice

ATDR	
num_events	number of events under inspection
arrivals	number of arrivals to the model
completions	number of departures from the model
Time	global inspection time
lamda	arrival rate
mu	service rate
q	Round-robin time slice
queueMaxLength	Maximum queue length
customerDropped	Total Customers dropped
idum	random number generator seed
Station	Sertvice Station Count
Ranked	The calendar
Rho (p)	Density (lamda/mu)
Handler()	Schedule arrivals
Enqueue()	Execute the simulation
Path()	Performance testing
Schedule()	Schedule customers

Model
- Customer::Handler(BeginMsg)
- Customer::Handler(CalendarMsg)
- Source::Handler(CalendarMsg)

Calendar
- Actor::Schedule()
- Simulate()

Station
- Station::Request()
- Station::Free()

Path
- Path::Up()
- Path::Down()
- Path::Mean()

Queues
- Queue::DequeuedItem()
- Queue::Empty()
- Queue::Size()
- FIFO::Enqueue()
- LIFO::Enqueue()
- Ranked::Enqueue()

Server Method

Packet arrives for processing, call function:
-> void Request(Actor * packet);
Once processed, call function to set server status as free:
-> void Free(void);

New Packet - Process or Queue Packet

```
void Station::Request(Actor * packet) {
BeginMsg beginmsg;
if (path->n < no_servers) // if a server is free
packet->Handler(beginmsg); // service the customer
else // if all servers are busy
wl->Enqueue(packet); // queue the packet
path->Up(Time); // compute the performance measures
}
```

Accept new packet

```
void Packet::Handler(BeginMsg &) {
double x;
x = Time + expdev(&tap_pkt_id)/mu;
Schedule(x); // schedule the completion event
}
```

Enqueue Method

```
class Queue {
public:
Queue(void) { first = NULL; };
~Queue(void) {};
virtual void Enqueue(Item * i) = 0;
Item * DequeuedItem(void);
int Size(void);
int Empty(void);
protected:
Item * first;
};
```

Round-robin Re-queueing

```
void Customer::Handler(BeginMsg &)
{
 float x;
  if(this->tm == 0)
  {
     this->tm = expdev(&idum)/mu;
     if(debug)
     {
       std::cout << "customer " << this->no << " : total = " << this->tm << "\n";
     }
  }
  x = Time + fmin(this->tm, q);
  Schedule(x); // schedule the next time slice event
  if(debug)
  {
     std::cout << "customer " << this->no << " : slice sheduled\n";
  }
}
```

Check for Queue Space, otherwise drop (Bypass)

```
if(wl->Size() < queueMaxLength){
wl->Enqueue(packet);// the server is busy, queue the customer
path->Up(Time);
}
else
{
if(debug)
{
std::cout << "Packet" << ((Packet*)packet)->no << "
dropped!\n";
}
packetsDropped++;
delete(packet);}}
```

Class performance path

```
void Path::Up(double T) {
W+= (double) (n) * (T - t);
n++;
t = T;
}
void Path::Down(double T) {
W+= (double) (n) * (T - t);
n--;
t = T;
}
double Path::Mean(double T) {
double x;
x = (W + (double) (n) * (T - t)) / T;
return(x);
}
```

Path Class: Performance Measurement

```
class Path {
public:
Path(void) { n = 0; W = t = 0.0; };
~Path(void) {};
void Up (double T);
void Down (double T);
double Mean(double T);
int n;
private:
double W, t;
};
```

Figure 3.1: High-level overview of ATDR process

3.1.1.2 Server and queue properties

Inter arrival time (IA) is the time between packet arrivals.The time to service a packet is known as the service time. IA times and arrival times are independent.
The arrival process can be defined as

$$A(t) = Pr(Xt) \tag{3.1}$$

Other queuing properties and their corresponding details can be seen in Table 3.1.

Table 3.1: Server and queue properties

Attribute	Details
Packets arriving	Can be similar or different
Arrival process	Avg. arrival rate λ Packets/Sec
Service process	Avg. Service Rate μ Packets/Sec
Queue backlog	Infinite or finite
Server service discipline	First In First Out (FIFO)
Number of servers	1 with multiple processing threads
Network priority requirements	Equal or different

Assumptions

- The IA times and the service times are independent.
- Successive IA times are independent.
- Successive service times are independent.
- The time to the next arrival is not dependent on the time since the last arrival and is considered memoryless.
- The only continuous distribution is the exponential distribution with memoryless (Markov) property.

3.1.1.3 Algorithm implementation

Arrival and Service processes

- THE ARRIVAL PROCESS | *IA time distribution*

$$A(t) = Pr(X \leq t) \tag{3.2}$$

- THE SERVICE PROCESS | *service time distribution*

$$B(t) = Pr(Y \leq t) \tag{3.3}$$

To determine the arrival rate (number of events per unit time), the server will function with the well-known Poisson process to model the random points in time and space. The Poisson distribution process, defined by λ, is a parameter that represents the expected number of events within a given interval or the highest probable number of events therein. Randomness mimics the number of k occurrences, with k being the number of completed occurrences within the given time slot. K is defined as $P(X = k) = e$ ^ (-μ) $* \mu\hat{}k/k!$.

Suppose no arrival occurred in seconds (s), then (t) is the remaining time until the next arrival. Thus, the distribution for (t) is given by:

$$Pr(X \leq s + t \| X \; s) = Pr(X \leq t) = 1 - e^{-} \mu t \tag{3.4}$$

3.1.1.4 Merging and splitting of Poisson processes

Merging function

Let $N_1(t)$ and $N_2(t)$ be two independent Poisson processes with rates λ_1 and λ_2 respectively. The random process N_t is obtained by combining the arrivals $N_1(t)$ and $N_2(t)$. Therefore, we claim that N_t is a Poisson process with rate λ = λ_1 + λ_2. Since N_1 and N_2 are independent and increases independently, we conclude that N_t also increase independently.

Let $N_1(t)$, $N_2(t)$, $\cdots$, $N_m(t)$ be m independent Poisson processes with rates λ_1 , λ_2, $\cdots$, λ_m. Let also, $N_1(t)$ = $N_1(t)$ + $N_2(t)$ + $\cdots$ + $N_m(t)$ for all t $\sum$ [0,∞]. Then, N_t is a Poisson process with rate λ_1 + λ_2 + $\cdots$ λ_m .

Splitting function

For packets arriving and intelligently forwarded, or packets steered to the upper layers of the server for fingerprint processing, suppose the number of packets arriving at the server for processing in a given time interval I is N ~ Poisson (λ). Assuming that each packet is treated independently from the other packets in the queue, and from the value of N, with a probability of p, let X be the number of packets processed in that time interval. Also, let Y be the number of packets that were not processed; such that $X + Y = N$. Thus, to split the Poisson process, let $N(t)$ be a Poisson process with rate λ. For the example, we will divide the process into two processes: $N_1(t)$ and $N_2(t)$. Upon each arrival, the queues will be analysed with $P(H) = p$. Arrivals are independent of each other and are independent of $N(t)$. Thereby, $N_1(t)$ is a Poisson process with rate λp, $N_2(t)$

is a Poisson process with rate $\lambda(1-p)$ and $N_1(t)$ is independent of $N_2(t)$. When a large number of statistically independent processes are merged, the merged process will be approximately Poisson.

3.1.1.5 Queuing function

Queue processing derivatives

- The average number of packets being serviced in the queue at any given time is given by:

$$\bar{W} = \frac{\sum_i n_i X(t_f - t_{f-1})}{t_f} = \frac{W + \sum_i n_i X(t_f - t_{f-1})}{t_f} \tag{3.5}$$

- Duration that the server occupies while processing the queue is given by:

$$\bar{W} = \sum_i n_i X(t_i - t_{i-1}) \tag{3.6}$$

- The number of packets in the queue at any time
- Running total amount of packets in the queue

Poisson process

The probability of an event occurring in a small interval of time $\triangle\, t$ is $\lambda \triangle\, t$
Probability of more than one event occurring in $\triangle t$ is zero.
Probability of n events occurring during time t is:

$$P_{\bar{n}}(t) = \frac{(\lambda t)^n e^- \lambda t}{n!} \tag{3.7}$$

If a Poisson process is formed by the events, the inter-events times are exponentially distributed:

$$A(t) = Pr(X \leq t) = \sum_{n=1}^{\infty} P_n(t)n = 1 - e^{-\lambda t} \tag{3.8}$$

M/M/1/N with memoryless property

Queue length distribution:

$$P_n = (1 - p)p^n \tag{3.9}$$

Note:

$$p_n = \lambda / \mu implies \lambda \;\; \mu \tag{3.10}$$

Average queue length:

$$Q = \sum_{n=1}^{\infty} nP_n = \frac{p}{1 - p} \tag{3.11}$$

Average waiting time:

$$W = Q/\lambda = \frac{1}{\mu(1 - p)} \tag{3.12}$$

3.1.2 Network packet processing using round-robin packet scheduling

Round-robin is a classic scheduling algorithm, which allocates equal service time to each queue. It has been expensively used in numerous domains, including operating systems, task scheduling, job shops and most importantly traffic optimisation [16, 17]. In this study, round robin is being used to optimise network traffic flow across multiple queues, as it is a fair scheduling algorithm, which ensures that each queue gets equal access to the server for equal amounts of time.

3.1.3 M/M/1/N round-robin queue | Fundamental time quantum election

$$\lambda \longrightarrow \textcircled{\mu} \supset\longrightarrow \tag{3.13}$$

M/M/1/N round-robin queue is an enhancement to the M/M/1/N queue, wherein each queue $n \in N$ gets equal service time with the single server (where N is the number of queues). The queue length distribution is given by:

$$P_n = (1 - p)P^n \tag{3.14}$$

3.2 Autonomous Threat Detection and Response (ATDR)

The proposed method is designed to detect multi-vulnerability attack vectors at line rate, by means of deep learning packet inspection on network traffic flow in real-time. The method consists of optimal formulations and functions focused on attack and detection of malicious activities by monitoring behaviours of threats. Zero-day attacks are known to be when vendors have not yet released patches to resolve vulnerabilities in their systems and appliances. Thus, the proposed real time system model will contribute toward protection against zero-day attacks through artificial intelligence. The goal of ATDR is to identify distinct patterns in a variety of noisy data sets, automatically identify and cluster traffic into categorical and accurate classes. [3] states that the accuracy of the classifiers should be competitive, equal to, or should surpass the accuracy achieved by classifiers manually created by human experts. This chapter presents the ATDR method shown in Figure 3.2, which improves the completeness, efficiency and accuracy of existing classification systems. ATDR also leverages on ML, incorporating feature selection, signature discovery and fingerprint production processes. These processes are classified as unsupervised ML clustering.

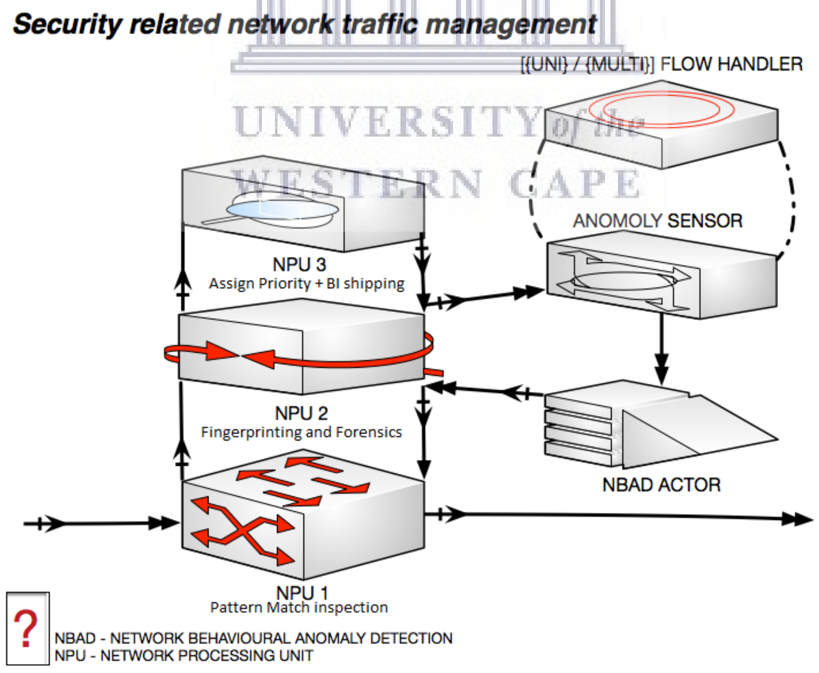

Security related network traffic management
[{UNI} / {MULTI}] FLOW HANDLER
ANOMOLY SENSOR
NPU 3
Assign Priority + BI shipping
NPU 2
Fingerprinting and Forensics
NBAD ACTOR
NPU 1
Pattern Match inspection
?
NBAD - NETWORK BEHAVIOURAL ANOMALY DETECTION
NPU - NETWORK PROCESSING UNIT

Figure 3.2: ATDR - Behavioural threat detection system

The behavioural threat hunting logic searches for an optimal (reduced) feature set to describe each data set. NPU1 (pattern match inspection) matches the derived feature set against known fingerprint patterns. In NPU2 (fingerprinting and forensics), new patterns, referred to as fingerprints in this research, are automatically identified and grouped (clustered) into these new data sets. NPU3 is the response process to detected threats. It discards anomalous and malicious activity packets and terminates any further processing for resource preservation.

3.2.1 Threat hunting process

Autonomous Threat Detect and Response functional layers consist of a processing layer, inspection layer, and production layer, executing above the network behavioural anomaly detection (NBAD) layer. The layers are explained below and the workflow is illustrated in Figure 3.2. Between each of the functional layers are sensors that serve as triggers for action traffic flow, switching between the layers depending on the state of classification. Sensors in the layers will also instantly initiate flow packet dropping if a malicious behavioural pattern is matched.

3.2.2 Multi-vector threat flow handler

Parallel server processes are dedicated to scrub the traffic flows on the lowest layer so that it can be shunted/blocked to reduce processing and resource overhead. The goal is not to cover the broad spectrum of traffic inspection for potential anomalous and malicious traffic, but rather to recognise popular and obvious malformed traffic in the shortest time. This is to enforce packets drop before being further processed by the system, network and devices.

3.2.3 Network Processing Unit 1 (NPU1) - Ingest traffic pattern match

NPU1 is a workload processing cluster consisting of multiple processing queues that combine intersection and processing capacity. It ensures that all packets associated with a flow and session are always processed by the same queue, regardless of the physical interface through which they enter the cluster. This is achieved by the ATDR hashing algorithm proposed in this paper.

3.2.3.1 Feature learning and selection

Authors in [3] describe a feature vector as an n-dimensional vector of numerical features that describe an object. ML algorithms identify future instances of an object within a mixed data set by the training of these vectors. Extra computational overheads are required for vectors with increased dimensionality and are more task intensive. Thus, it is best to discover a more optimised feature set for each problem.

Feature selection is performed at the NPU1 layer, where the ATDR method, as illustrated in Figure 3.3, also encodes each feature subset by first projecting the feature into a low-dimensional subspace. This is then followed by a normalisation step to obtain a compact binary vector. Locality sensitive hashing (LSH) and its extensions are based on randomised projections and are one of the most widely employed hashing methods in ANN-related literature [5].

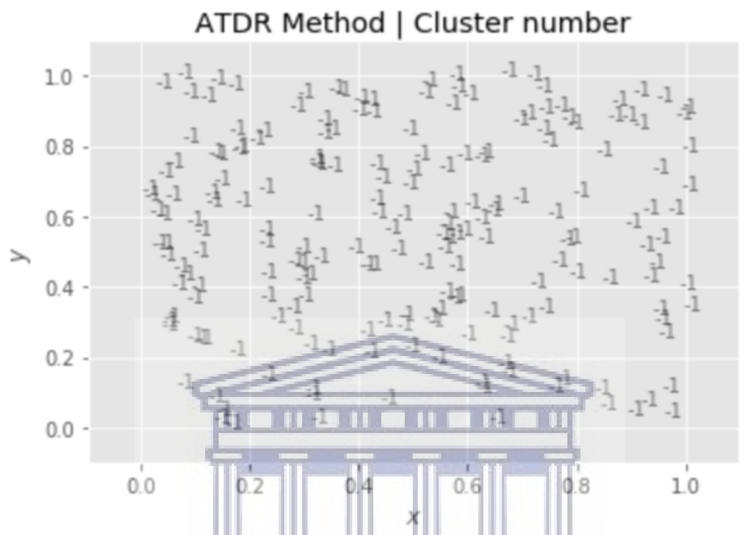


Figure 3.3: Normalising features to 1 or 0

The functions of NPU1 are as follows:

- Pattern signature cross hashing, which reduces processing overhead
- Packet and process queue hash mapping
- Mapping to known signatures
- Elevating unknown patterns to upper layer NPU2 for fingerprinting and forensics

3.2.3.2 NPU1 network layer function

This is a function in the first layer where traffic is captured, and deep packet inspection performed with the proposed processing architecture. A sensor switches the traffic to the second layer for intelligent processing.

3.2.3.3 NPU1 situational awareness function

Here the Naive Bayes ML method will be applied to learn and match fingerprints and assign to the closest network traffic protocol type as revealed by the Bayes network classification. The system will be an unsupervised ML expert system infused with the k-means algorithm for pattern recognition, decisions and reasoning executions.

3.2.4 Network Processing Unit 2 (NPU2) - Fingerprinting and forensics

NPU2 is a second layer of the workload processing cluster, dedicated to learning and formulating a baseline and traffic profiling for fingerprinting and/or forensics. This layer steers the traffic to a parallel NBAD sensor process which compares the traffic against known malicious behavioural fingerprints. If a malicious match is found, the traffic is shunted to the NBAD actor which enforces block rules that include instant black-holing. If no malicious matches are found, the traffic is forwarded to the NPU3 layer for further learning.

The functions of NPU2 are as follows:

- Arriving packets are forwarded to the uni- and multi-flow handling anomaly sensor and NBAD actor to scrub the traffic flows for anomalous and abnormal occurrences. If there are no exact known threats, traffic is matched against known fingerprint signatures or forwarded to NPU3 for deeper analysis.
- Anomaly sensor tags the flows if malicious activities are matched.
- NBAD actor drops the flows upon anomaly sensor tagging.

3.2.4.1 Traffic flow fingerprinting

According to [3], human experts are often used to annotate each datum with its associated class label. This process is inefficient, prone to error, time-consuming and labour-intensive. This further justifies the need for ATDR, as it is capable of grouping feature vectors of a particular type (pattern) using unsupervised learning algorithms.

3.2.5 Network Processing Unit 3 (NPU3) - Deep learning and traffic clustering

This layer involves DL cluster building and enhancement. From NPU2, the sensors switch the unknown traffic flows to NPU3 which performs detailed analytics to learn and automatically enhance cluster definitions by means of DL clustering techniques. If a match is found, traffic flow is forwarded with the applicable traffic protocol family, inheriting the associated priorities.

The roles of NPU3 are as follows:

- It groups unique traffic characteristics into small clusters for granular control.
- It assigns traffic to high or low VP group priority based on traffic classes.

3.2.5.1 Traffic pattern recognition (PR) and clustering

In pattern recognition, when the number of clusters is unknown, algorithms such as Density-Based Spatial Clustering of Applications with Noise (DBSCAN) and k-means provide effective means to infer these automatically. For the k-means algorithm, the stopping criteria is met when there are no more changes detected at the centroid locations.

3.2.5.2 Feature selection from data set

Table 3.2 summarises the paramount features required from the data set.

Table 3.2: ATDR - Feature datum of an IP packet

Feature	Description
Packet Direction	The direction of packets within a flow (source to destination, destination to source.
Packet Length	Packet length statistics (min, max, quartiles, standard deviation).
Packet IAT	Packet inter-arrival time (min, max, quartiles, standard deviation).
Flow Duration	The duration of a particular flow on the network, from its start to termination.
Number of Packets	The total number of packets observed for a particular flow.

3.3 Overview of network attacks

Authors in [18] proved that the common goal of cyber attacks is to affect the service availability for legitimate traffic. This is performed by sending malformed packets or by overloading the resources with excessive requests that are aimed to exhaust the resource capacity required to serve legitimate requests. The authors further divided attacks into three types namely: 1) volume-based attacks, 2) protocol-based attacks and 3) application-layer-based attacks. Volume-based attacks are often performed through botnets (millions of infected systems) of multiple Trojan infected systems and one system is made the controller of that botnet. The controller signals the botnet to perform attack at scale against a server and/or service [18]. Protocol-based attacks launch a flood of requests to a port service, for which the server allocates resources to serve each request, cumulatively exceeding the total resource capacity of the target. Application-based attacks are similar, in that a flood of application layer requests are sent to an application for processing until the queue in requests overwhelms the application's ability to function reliably [18]. ICMP flood attacks are when attackers misuse the administration ping service, a service used to check if hosts are alive, by sending millions or billions of ping requests until the server stops responding [18]. In SYN flooding, the attacker sends SYN packets to the server from spoofed IP addresses. The server tries to respond to each by sending the SYN/ACK packet, but as the fake IP does not respond, the server awaits multiple open replies and wastes resources for each request. The attacker further sends an additional flood of requests so that the server may stop responding to the legitimate users. In UDP floods, the server sends ICMP replies to a flood of UDP requests from a spoofed IP address, thus causing the system to be unresponsive and unavailable for the legitimate users [18]. DNS amplification is a technique where the attacker uses the DNS open resolver and the victims IP. The attacker sends the victim's IP through a botnet to open recursive servers; thus, the servers send a large number of packets overwhelming the victim [18]. Zero-day attack threats are explained as cases when the vulnerability has not yet been detected by anyone and no patches for the vulnerability have been released by the vendors. The attacker uses this window and exploits the vulnerability by crafting the packet and sending to the victim to cause the system to crash.

3.3.1 Network behavioural anomalies

This subsection briefly describes various types of network behavioural anomalies considered in the model [19]:

- ***Invalid Packets***: Packet inspection for malformed IP headers, short packets, invalid TCP flags, invalid ACK numbers and bad IP, TCP and UDP checksum.
- ***NX Domain per Second***: Benchmarking the baseline rate for NX domain responses, and enforcing actions if any influx is detected.
- ***DNS Rate Limiting***: Benchmarking the baseline flow rate of DNS requests
- ***DNS Malformed***: DNS packets that do not conform to the standard RFC rules

- ***TCP Flow Resets***: Packet inspection for TCP resets, considering the source to destinations counts
- ***TCP Out-of-sequence Authentications***: Benchmarking the baseline flow rate of SYN authentications to the internal segments and taking influx if a burst in rates is detected
- ***ICMP Floods***: The originating traffic source port will be baselined and monitored for any spikes in rate, while also considering the size of the ping packets and the counts of source to destination hosts
- ***UDP Floods***: According to [20], UDP flood attacks send an enormous number of UDP packets to any random ports of a server to impact other legitimate traffic to the network port. UDP's flooding technique is to exhaust the target machine with a large number of UDP packets. The network bandwidth continues to degrade until services malfunction. Trending of 95th percentile traffic packet baselines will be learned over intervals of a day, one week and finally one month, to forecast and match any rapid or abnormal packet bursts to build a confidence in behavioural variance in comparison to normal expected traffic patterns.
- ***Session Attacks***: Exhaust a server's resources through empty sessions, resulting in complete system malfunction or service-impacting system performance. These attacks can bypass defence mechanisms that only monitor incoming traffic on the network.
- ***SYN Floods***: In order to respond against SYN floods, the need to be detected by monitoring that the TCP three-way handshakes are established correctly. In normal behaviour: client sends a SYN message to the server, the server responds with a SYN-ACK and then the client responds with an ACK. The three steps above establish a session correctly. Monitoring the rates of the new session handshake steps could reveal attempts to intercept or hijack sessions and/or attempt to flood the server with high loads of negotiations around the connection state.

3.3.2 Malware database pattern identification with external attack and exploit feeds

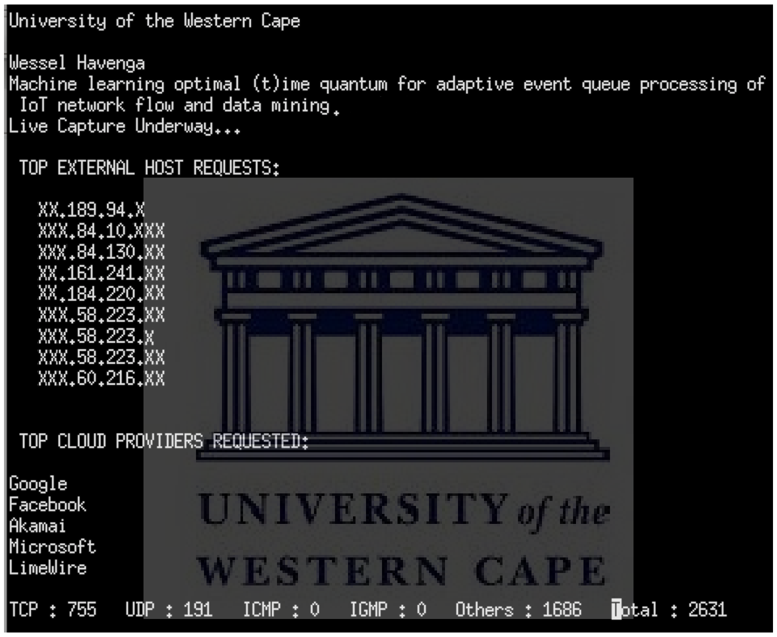
Malware databases are released by anti-virus companies and research organisations to enable researching of algorithms. The following sources are considered in this study:
• http://www.netresec.com/?page=PcapFiles
• https://wiki.wireshark.org/SampleCaptures

• https://snap.stanford.edu/data/#email

• http://www.secrepo.com/

3.4 Use cases implementation

For the implementation phase, simulations were conducted at internal and external network points to obtain a broad collection of traffic behavioural patterns, similar to Figure 3.4.



```
University of the Western Cape

Wessel Havenga
Machine learning optimal (t)ime quantum for adaptive event queue processing of
 IoT network flow and data mining.
Live Capture Underway...

 TOP EXTERNAL HOST REQUESTS:

   XX.189.94.X
   XXX.84.10.XXX
   XXX.84.130.XX
   XX.161.241.XX
   XX.184.220.XX
   XXX.58.223.XX
   XXX.58.223.X
   XXX.58.223.XX
   XXX.60.216.XX


 TOP CLOUD PROVIDERS REQUESTED:

Google
Facebook
Akamai
Microsoft
LimeWire

TCP : 755   UDP : 191   ICMP : 0   IGMP : 0   Others : 1686   Total : 2631
```

Figure 3.4: Capturing real-time traffic

Once captured, the packets were anonymised, as required for data privacy. Anonymisation was achieved by randomising IP addresses, ports and payloads of traffic records used for statistical analysis to prevent leakage of personally identifiable data. The authors in [21] contributed an effective variant of C++ code that modifies packet headers to remove link-layer addresses used in anonymising IP addresses, and shuffles UDP/TCP ports. Crypto_Pan is also a cryptographic algorithm that can be applied for anonymising IP addresses, while preserving the subnet structure. Any string of bits x is encrypted to a new string $Ek(x)$ while ensuring that for any bit-stream pairs (x,y), that share a common prefix of length p, their images $Ek(x), e, phEk(y)$ also share a common prefix length p. Bit-strings of length n are encrypted by Crypto_Pan by descending a binary tree of depth n one step for each bit in the string. A pseudo-random function assigns a “0” or “1” to each of the binary tree's $2n - 1$ non-leaf nodes.

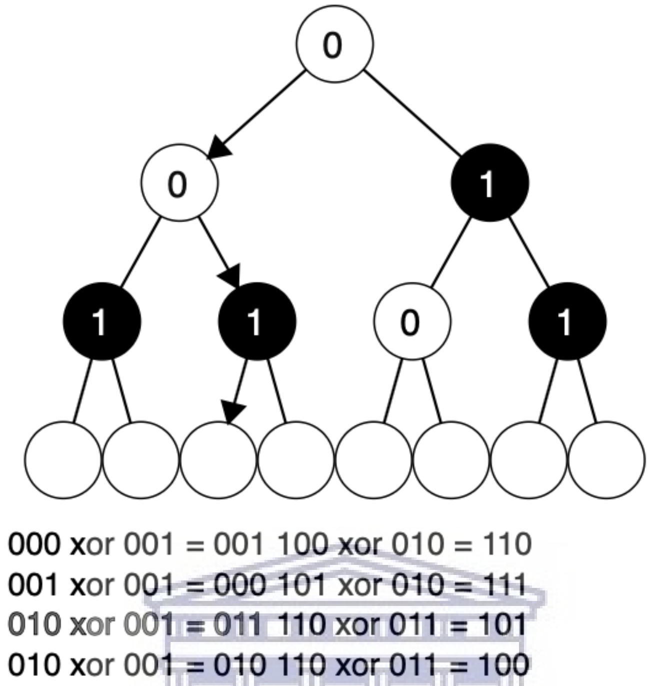

0
0
1
1
1
0
1
000 xor 001 = 001 100 xor 010 = 110
001 xor 001 = 000 101 xor 010 = 111
010 xor 001 = 011 110 xor 011 = 101
010 xor 001 = 010 110 xor 011 = 100

Figure 3.5: Crypto_Pan algorithm as a tree descent

In the Crypto_Pan figure above, the tree nodes are coloured according to a pseudo-random function of the key material. The arrows show the descent corresponding to the input bit-string "010". That descent's nodes are coloured "001" and, thus, the anonymised output string is "011".

3.4.1 IP & port anonymisation

The algorithm used for IP anonymisation in this study is based on Crypto_Pan with features that ensure:

- prefixes are preserved
- one-to-one mapping
- IP anonymisations are reproducible and reversible via the use of a consistent 256-bit key
- port anonymisations are reproducible and reversible via the use of a consistent 16-bit seed
- port numbers are anonymised to a different port number
- AES encryption and the mappings are pseudo-random.

3.4.2 Traffic classification and clustering

3.4.2.1 Data pre-processing

Network TCP flow data combined with payload bytes for every TCP session are collected. Each byte is represented by an integer range of 0 to 255, which are then normalised to between 0 and 1. The length of each payload sequence is set to 1,000. Finally duplicate data are discarded. For data visualisation, each payload is depicted with a picture where each byte is represented as a pixel.

3.4.2.2 Clustering – DBSCAN threat detection classifier

DBSCAN is an n-dimensional clustering algorithm that provides benefits over other clustering algorithms in that: 1) clusters can be non-spherical, 2) it can differentiate between clustered end-points and noise points and 3) it does not require the number of points to be specified in advance. Data clustering performance tests were performed against random generated data sets according to the best values for hyper-parameters for optimal clustering as derived from the search by [3] with eps of 0.1039 and a minPTS of 152.

Jupyter Notebook is an open-source web application that provides the ability to create and share live code documents, equations, narrative tests and visualisations toward use cases that include numerical simulation, statistical modelling, data visualisation, machine learning and others. The combination of Jupyter Notebooks and DBSCAN algorithmic code sets provided the means to perform clustering performance tests and the comparison of the output results with the optimal findings of other authors.

Chapter 4

PERFORMANCE EVALUATION

4.1 Performance evaluation overview

When traffic passes through the inspection sensors, the sensors execute real-time threat hunting: fingerprinting and matching for categorisation of the traffic flows. As one of the response functions of ATDR, it tags and drops anomalous and malformed network flow packets. Thereafter, it forwards the clean traffic in a supervised learning manner. For the detection role of ATDR, traffic flow profile inspections and learning are achieved by means of deep packet inspection as traffic passes through the ATDR system sensors. To prevent depletion of the sensor servers' resources, malicious traffic is inspected and removed by network behavioural signature detector. This reduces queuing and provides a more secure and optimised infrastructure. System security and network management largely depends on network traffic classification according to [13]. The traffic inspection and classification model used in this study is based on Bayesian Networks and Naive Bayes. According to [22], these well-understood supervised learning algorithms have a classification approach that is based on probabilistic knowledge. Also, similar to [13], this model consists of various processing layers.

In addition to the response and detection functions of ATDR, each cluster group is automatically annotated with the coined term *lane priority* that produces training sets for the production of supervised and semi-supervised classifiers. In this study, the number of clusters is set explicitly or inferred automatically through a discovery and nomination process. Similar to the APIC method, where the number of fingerprints, patterns or targets are not specified as described in [3], this method used in this study utilises algorithms such as Density-Based Spatial Clustering of Applications with Noise (DBSCAN) to automatically determine number of clusters. That is, the number of targets, k, is known prior. Subsequently, an algorithm such as k-means is then used to group traffic into k clusters [1].

Once the server has processed the traffic and traffic flows are divided into clusters, each cluster is allocated a weighted priority. The main objective is to give traffic with highest priority expedited access. In this context, lane priority can be defined as the assignment of a weighted priority to traffic flows on a network path that has been divided into virtual protocol group (VP-group) while in transit. Lane priority assignment is a ranked numbering system representing a priority level and can be customised. In this study, four levels were used for testing; however, more levels can be allocated to enable more granular control and management as shown on Table 4.1. On the table, 'reservation' represents the maximum head-room in percentage allowed for each priority level. This allows for additional guarantees based on well-known committed information rate (CIR) and peak information rate (PIR). At every instant, the capacity available is divided among the flows considering the multidimensional combination of the flow demands, lane priority and reservation allocated.

Table 4.1: ATDR virtual cluster groups - Traffic lane prioritisation

Lane Priority	**Cluster**	**Protocol Fingerprints**	**Reservation**
1	P2P	BitTorrent, Kazaa	2
2	**TCP-ATTACK**	SYN-flood, Session-flood, TCP-ACK flood, Out-of-sequence packets	0
3	UPDATES	Microsoft, Apple, Android	8
4	WEB	HTTP, HTTPS, FTP, Email	20
5	**UDP-ATTACK**	NTP-flood, DNS-attack	0
6	VIDEO	YouTube, Skype, Gtalk,Webex	70

The rest of this chapter gives performance evaluation details of the proposed ATDR model and also discusses obtained results.

4.2 ATDR - Clustering

The clustering process is executed in a ten step process, which is illustrated in Figure 4.1 and enumerated in the list below:

- Step 1 - Live inspection and/or collection of traffic data.

- Step 2 - Introduction of offensive/threat traffic patterns.
- Step 3 - Cast the traffic data to a data frame training set.
- Step 4 - Data frame validation.
- Step 5 - Training data analytic.
- Step 6 - Collection and graphing of traffic analytic.
- Step 7 - Collection of data features sets using various algorithms.
- Step 8 - Feature grouping and learning.
- Step 9 - Calculate and discover the optimal eps and MinPTS for DBSCAN
- Step 10 - Cluster formation for further security-related traffic management.

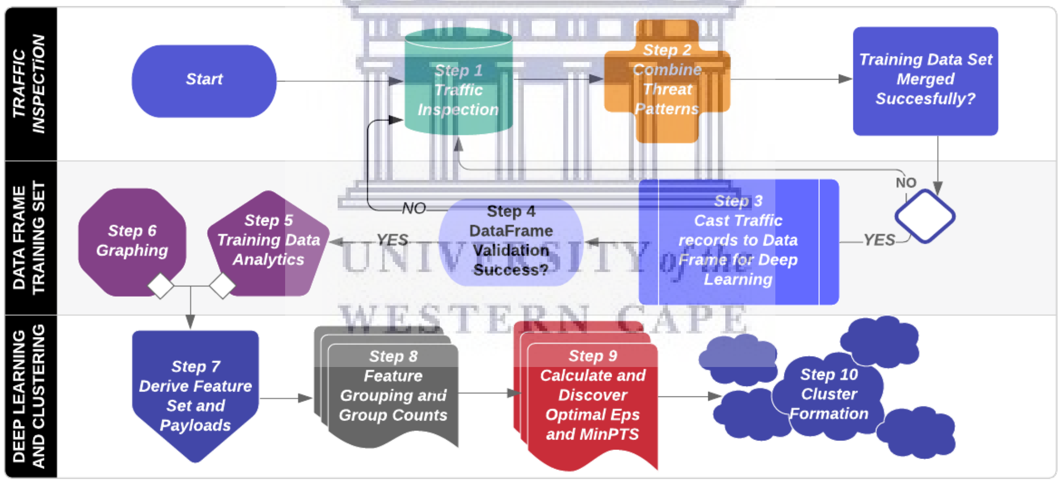

TRAFFIC INSPECTION
DATA FRAME TRAINING SET
DEEP LEARNING AND CLUSTERING
Start
Step 1 Traffic Inspection
Step 2 Combine Threat Patterns
Training Data Set Merged Succesfully?
NO
YES
Step 3 Cast Traffic records to Data Frame for Deep Learning
Step 4 DataFrame Validation Success?
NO
YES
Step 5 Training Data Analytics
Step 6 Graphing
Step 7 Derive Feature Set and Payloads
Step 8 Feature Grouping and Group Counts
Step 9 Calculate and Discover Optimal Eps and MinPTS
Step 10 Cluster Formation

Figure 4.1: ATDR ten step process

4.2.1 Step 1 | ATDR - Live inspection and collection of traffic data

In this step, network traffic is captured and inspected as it flows in real time. Figure 4.2 shows a snippet, wherein various packet captures are inspected to determine the response to the various legitimate and/or anomalous communication behaviours.

```
num_of_packets_to_sniff = 10
pcap = sniff(count=num_of_packets_to_sniff)
pcaplen = len(pcap)

# rdpcap returns packet list and packet list object can be enumerated
print(type(pcap))
print(len(pcap))
print(pcap)
pcap[0]
```

Figure 4.2: ATDR | Step 1 | Live capture

Captured traffic packets are then filtered using different parameters to build specific data sets that can be tested against various use cases.

```
<Sniffed: TCP:4 UDP:2 ICMP:0 Other:4>
<Ether  dst=ff:ff:ff:ff:ff:ff src=14:49:e0:68:87:9a type=ARP |<ARP  hwtype=0x1
ptype=IPv4 hwlen=6 plen=4 op=who-has hwsrc=14:49:e0:68:87:9a psrc=192.168.1.142
hwdst=00:00:00:00:00:00 pdst=192.168.1.1 |<Padding
load='\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'
|>>>
```

4.2.2 Step 2 | ATDR - Introduction of offensive traffic patterns

This step tests ATDR's ability to detect known threats. Multiple vector attacks were executed and mixed with the original (clean) traffic. These were processed by the system to test the ability to accurately separate different kinds of anomalous traffic heuristics and also differentiate legitimate traffic. Actual TCP reset, MS Blaster worm and DOS attacks are performed and captured for analysis.

The following are snippets of some of the captured traffic stream.

```
# ATDR - Multiple-vector attacks launched | 192.168.21.21 ----( D o S )----> 192.
  ↪168.21.22
pcap = rdpcap("pcap/attack-dos-tcp-rst-attack.pcap") # TCP RESET Attack Simulated
pcap = pcap + rdpcap("pcap/attack-dos_tcp_vertical_scans.pcap") # TCP Vertical␣
  ↪Port Scans Attack Simulated
pcap = pcap + rdpcap("pcap/attack-dos_tcp_syn_flood.pcap") # TCP SYN Flood␣
  ↪Attack Simulated
pcap = pcap + rdpcap("pcap/attack-msblaster_worm_attack.pcap") # MS Blaster worm␣
  ↪exploitation Attack Simulated

<attack-dos-tcp-rst-attack.pcap+attack-dos_tcp_vertical_scans.pcap+attack-
dos_tcp_syn_flood.pcap+attack-msblaster_worm_attack.pcap: TCP:2792 UDP:1035
ICMP:89 Other:0>
```

ATDR - TCP segment structure

Each encapsulated layer of frames/packets/segments consists of fields that ATDR extracts from the packet list. Layer 3 (IP) and Layer 4 (TCP/UDP) are inspected with payload in the following sequence: ETHERNET -> Internet Protocol -> Layer 4 segments.

The ATDR feature datum below reflects the payload that ATDR inspects to calculate the hashes for traffic clustering.

```
[ATDR Feature Datum]
///////////////////////////
Packet length:  40
Packet payload:
b'zi\x00\x19bJ\xf2\x0f\x16\xb3\x15\x80P\x04\x02\x00\x07U\x00\x00'
Destination Port:  25
///////////////////////////

Ether / IP / TCP 192.168.21.22:31337 > 192.168.21.21:smtp R
IP / TCP 192.168.21.22:31337 > 192.168.21.21:smtp R
TCP 192.168.21.22:31337 > 192.168.21.21:smtp R
```

```
###[ Ethernet ]###
  dst       = 00:0c:29:4d:34:fd
  src       = 00:0c:29:3f:0e:13
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 40
     id        = 37086
     flags     =
     frag      = 0
     ttl       = 42
     proto     = tcp
     chksum    = 0x5476
     src       = 192.168.21.22
     dst       = 192.168.21.21
     \options   \
###[ TCP ]###
        sport     = 31337
        dport     = smtp
        seq       = 1649078799
        ack       = 380835200
        dataofs   = 5
        reserved  = 0
        flags     = R
        window    = 512
        chksum    = 0x755
        urgptr    = 0
        options    = []
```

ATDR - Training data set statistics

Profiling network communication produces intelligence around the different traffic heuristics. Traffic fingerprint profiles are obtained through a combination of similar characteristics for both legitimate protocol traffic and distinct patterns associated to the anomalous traffic as derived from the simulated attack training sets.

```
Ethernet <Ether from attack-dos-tcp-rst-attack.pcap+attack-
dos_tcp_vertical_scans.pcap+attack-dos_tcp_syn_flood.pcap+attack-
msblaster_worm_attack.pcap:
TCP:2792 UDP:1035 ICMP:89 Other:0>
IP <IP from attack-dos-tcp-rst-attack.pcap+attack-
dos_tcp_vertical_scans.pcap+attack-dos_tcp_syn_flood.pcap+attack-
msblaster_worm_attack.pcap:
TCP:2792 UDP:1035 ICMP:89 Other:0>
TCP <TCP from attack-dos-tcp-rst-attack.pcap+attack-
dos_tcp_vertical_scans.pcap+attack-dos_tcp_syn_flood.pcap+attack-
msblaster_worm_attack.pcap:
TCP:2792 UDP:0 ICMP:0 Other:0>
UDP <UDP from attack-dos-tcp-rst-attack.pcap+attack-
dos_tcp_vertical_scans.pcap+attack-dos_tcp_syn_flood.pcap+attack-
msblaster_worm_attack.pcap:
TCP:0 UDP:1035 ICMP:0 Other:0>
```

4.2.3 Step 3 - Cast traffic to training data set

In Step 3, various types of malicious traffic and attacks were generated and then captured to serve as the data set, as shown in the snippet below.

```
# Collect field names from IP/TCP/UDP (These will be columns in the Dataframe)
ip_fields = [field.name for field in IP().fields_desc]
tcp_fields = [field.name for field in TCP().fields_desc]
```

```
udp_fields = [field.name for field in UDP().fields_desc]

dataframe_fields = ip_fields + ['time'] + tcp_fields +␣
 ↪['payload','payload_raw','payload_hex']
```

4.2.4 Step 4 - ATDR data frame validation

At this stage, the traffic flow packet fields are inspected to perform classification of protocol family types and also anomolous packets.

```
# Retrieve first row from DataFrame
# Return first 5 rows
# Return last 5 rows
# Return the Source Address for all rows
# Return Src Address, Dst Address, Src Port, Dst Port
```

```


IP Packet fields extracted to derive ATDR features

version                 4
ihl                     5
tos                     0
len                    40
id                  33862
flags
frag                    0
ttl                    42
proto                   6
chksum              24846
src          192.168.21.22
dst          192.168.21.21
options                 0
```

```
time           1570563493.006009
sport                      31337
dport                         25
seq                   1269542725
ack                   1466347216
dataofs                        5
reserved                       0
flags                          R
window                       512
chksum                     34490
urgptr                         0
options                     None
payload                        0
payload_raw                  b''
payload_hex                  b''
Name: 0, dtype: object
(3916, 28)


Source to Destination Ports:


                  src             dst  sport  dport
0        192.168.21.22   192.168.21.21  31337     25
1        192.168.21.22   192.168.21.21  31337     25
2        192.168.21.22   192.168.21.21  31337     25
3        192.168.21.22   192.168.21.21  31337     25
4       192.168.21.254  192.168.21.100  33576  37008
...                ...             ...    ...    ...
3911     192.168.21.22   192.168.21.21  31337    135
3912     192.168.21.22   192.168.21.21  31337    135
3913     192.168.21.22   192.168.21.21  31337    135
3914     192.168.21.21   192.168.21.22   None   None
```

```
3915   192.168.21.22   192.168.21.21  31337    135
```

4.2.5 Step 5 | ATDR - Training data set Analytics

The traffic flows were analysed and some of the obtained outputs are shown in the snippet below. These include: the source and destination IP addresses, and the most active source and destination ports.

```
# Top Source address
count              3916
unique                3
top       192.168.21.22
freq               2793
Name: src, dtype: object

# Top Destination address
count              3916
unique                3
top       192.168.21.21
freq               2793
Name: dst, dtype: object



# Who is the top address speaking to?
['192.168.21.21']



# Who is the top address speaking to (Destination Ports)
[25 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 26 27
 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51
 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75
 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99
 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117
 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135
 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153
 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171
 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189
 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207
 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225
 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243
 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261
 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279
 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297
 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315
 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333
 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351
 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369
 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387
 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405
 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423
 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441
 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459
 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477
 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495
 496 497 498 499 500 501 502 503 504]


# Who is the top address speaking to (Source Ports)
[31337 2211 2212 2213 2214 2215 2216 2217 2218 2219 2220 2221 2222 2223
 2224 2225 2226 2227 2228 2229 2230 2231 2232 2233 2234 2235 2236 2237
 2238 2239 2240 2241 2242 2243 2244 2245 2246 2247 2248 2249 2250 2251
```

```
2252 2253 2254 2255 2256 2257 2258 2259 2260 2261 2262 2263 2264 2265
2266 2267 2268 2269 2270 2271 2272 2273 2274 2275 2276 2277 2278 2279
2280 2281 2282 2283 2284 2285 2286 2287 2288 2289 2290 2291 2292 2293
2294 2295 2296 2297 2298 2299 2300 2301 2302 2303 2304 2305 2306 2307
2308 2309 2310 2311 2312 2313 2314 2315 2316 2317 2318 2319 2320 2321
2322 2323 2324 2325 2326 2327 2328 2329 2330 2331 2332 2333 2334 2335
2336 2337 2338 2339 2340 2341 2342 2343 2344 2345 2346 2347 2348 2349
2350 2351 2352 2353 2354 2355 2356 2357 2358 2359 2360 2361 2362 2363
2364 2365 2366 2367 2368 2369 2370 2371 2372 2373 2374 2375 2376 2377
2378 2379 2380 2381 2382 2383 2384 2385 2386 2387 2388 2389 2390 2391
2392 2393 2394 2395 2396 2397 2398 2399 2400 2401 2402 2403 2404 2405
2406 2407 2408 2409 2410 2411 2412 2413 2414 2415 2416 2417 2418 2419
2420 2421 2422 2423 2424 2425 2426 2427 2428 2429 2430 2431 2432 2433
2434 2435 2436 2437 2438 2439 2440 2441 2442 2443 2444 2445 2446 2447
2448 2449 2450 2451 2452 2453 2454 2455 2456 2457 2458 2459 2460 2461
2462 2463 2464 2465 2466 2467 2468 2469 2470 2471 2472 2473 2474 2475
2476 2477 2478 2479 2480 2481 2482 2483 2484 2485 2486 2487 2488 2489
2490 2491 2492 2493 2494 2495 2496 2497 2498 2499 2500 2501 2502 2503
2504 2505 2506 2507 2508 2509 2510 2511 2512 2513 2514 2515 2516 2517
2518 2519 2520 2521 2522 2523 2524 2525 2526 2527 2528 2529 2530 2531
2532 2533 2534 2535 2536 2537 2538 2539 2540 2541 2542 2543 2544 2545
2546 2547 2548 2549 2550 2551 2552 2553 2554 2555 2556 2557 2558 2559
2560 2561 2562 2563 2564 2565 2566 2567 2568 2569 2570 2571 2572 2573
2574 2575 2576 2577 2578 2579 2580 2581 2582 2583 2584 2585 2586 2587
2588 2589 2590 2591 2592 2593 2594 2595 2596 2597 2598 2599 2600 2601
2602 2603 2604 2605 2606 2607 2608 2609 2610 2611 2612 2613 2614 2615
2616 2617 2618 2619 2620 2621 2622 2623 2624 2625 2626 2627 2628 2629
2630 2631 2632 2633 2634 2635 2636 2637 2638 2639 2640 2641 2642 2643
2644 2645 2646 2647 2648 2649 2650 2651 2652 2653 2654 2655 2656 2657
2658 2659 2660 2661 2662 2663 2664 2665 2666 2667 2668 2669 2670 2671
2672 2673 2674 2675 2676 2677 2678 2679 2680 2681 2682 2683 2684 2685
```

```
2686 2687 2688 2689 2690 2691 2692 2693 2694 2695 2696 2697 2698 2699
2700 2701 2702 2703 2704 2705 2706 2707 2708 2709 2710 2711 2712 2713
2714]
```

```
# Unique Source Addresses


# Unique Destination Addresses


Unique Source Addresses
['192.168.21.22' '192.168.21.254' '192.168.21.21']


Unique Destination Addresses
['192.168.21.21' '192.168.21.100' '192.168.21.22']
```

4.2.6 Step 6 | ATDR - Traffic intelligence

In this step, analytical results are displayed in graphical form. These aid in immediately "seeing" the unearthed patterns.

Figure 4.4 shows that the most active destination address was ’192.168.21.21’, while in Figure 4.5it can be seen that the three major source of traffic were ’216.58.223.131’, followed by ’192.168.1.107’ and ’172.217.170.68’.

```
# Group by destination address and payload sum
```

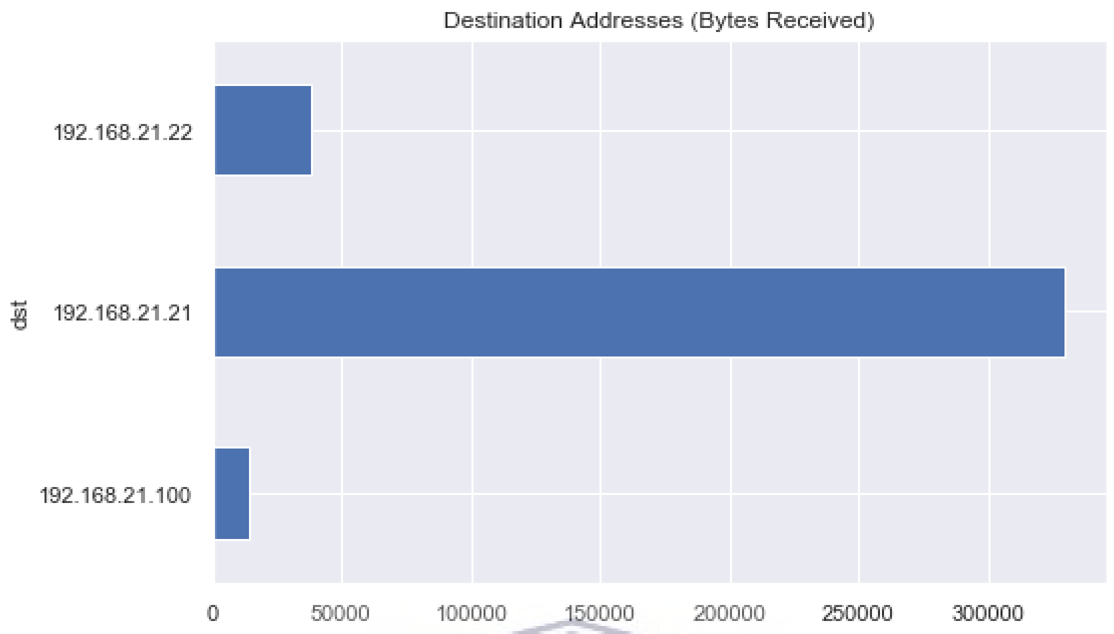


Figure 4.3: Group by destination address and payload sum

```
Store ATDR cluster feature set
ATDR_feature

dport
1            0
2            0
3            0
4            0
5            0
          ...
2700         0
2701         0
2702         0
31337        0
37008    13966
```

```
# Group by source address and payload sum
```

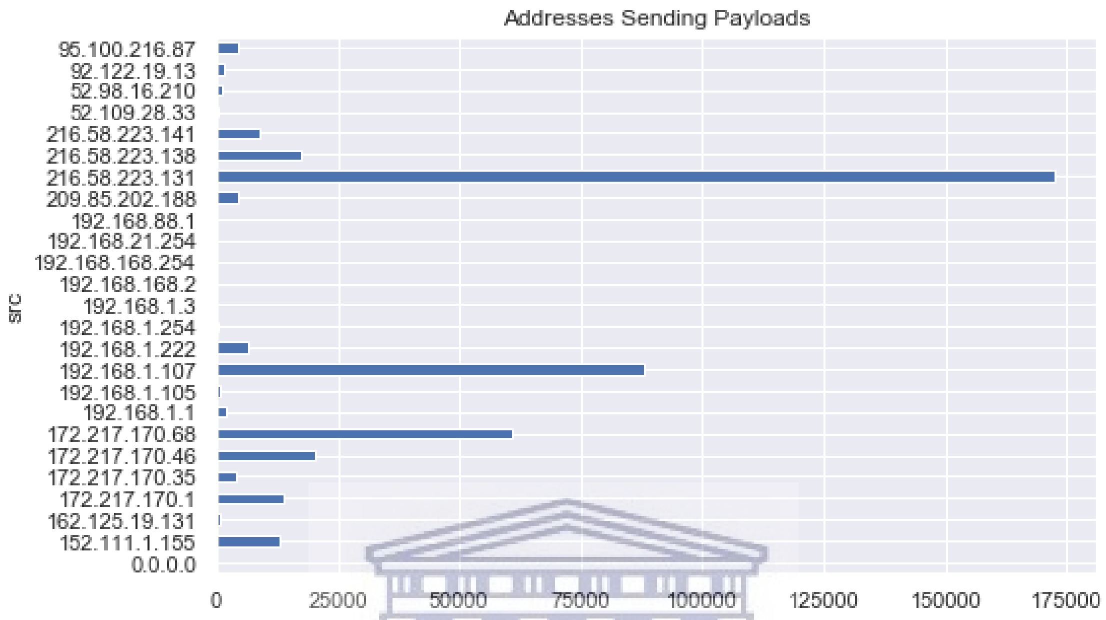


Figure 4.4: Group by source address and payload sum

```
# Group by destination address and payload sum
```

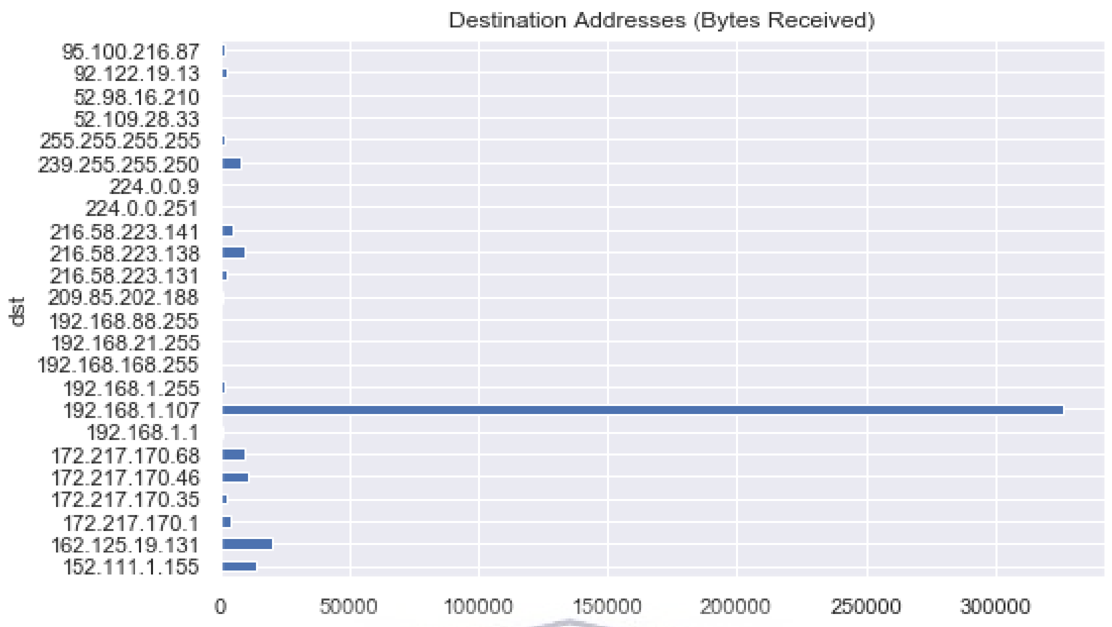


Figure 4.5: Results

4.2.7 Step 7 | ATDR - Payload Investigation

Having identified the most active data source, this step further analyses the "conversation" to identify which address has exchanged the most amount of bytes with most active source IP address. We can then create a data frame with only conversation from the most active and/or and suspicious addresses.

The obtained results are shown in the snippet below and Figure 4.6.

```
# Create data frame with only conversation from most frequent address
# Display Source address, Destination address, and group by payload only
# Plot the frequent address communication (by payload)


192.168.21.21 is detected as a potential victim
```

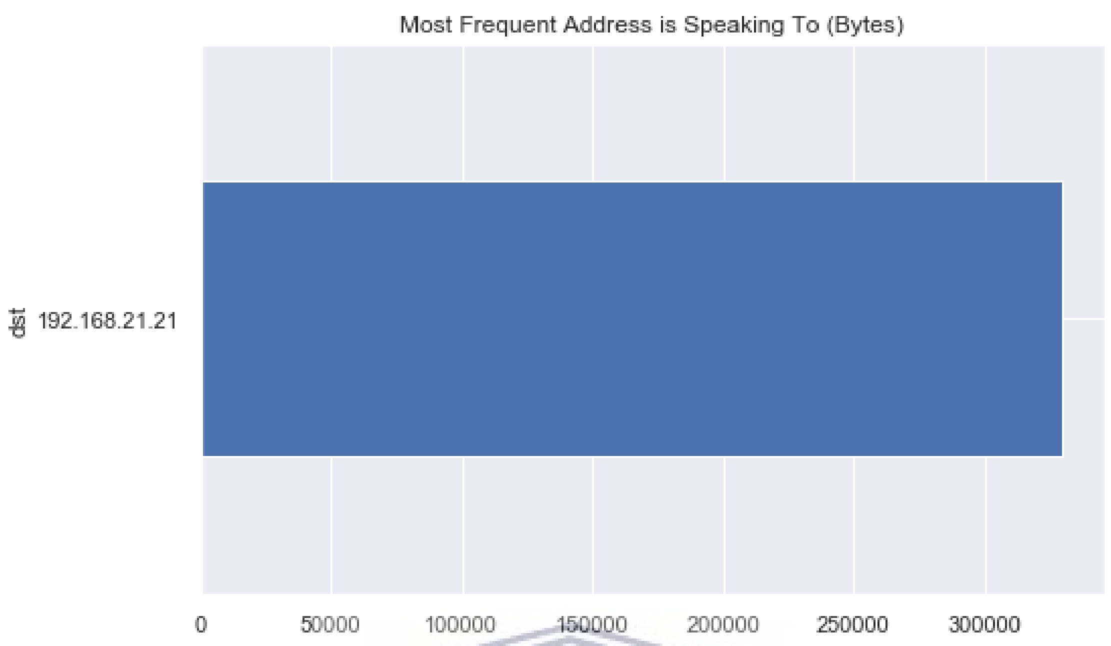


Figure 4.6: Payload validation

```
# Store each payload in an array
```

4.2.8 Step 8 | ATDR - Number of clusters

In this step, the total number of groups as clustering input is shown. A total of 998 groups were identified by ATDR and are depicted in Figure 4.7.

```
# Count of groups to input for Clustering
ATDR_clusters


998
```

```
#Cluster formation: learn to form accurate cluster groups
```

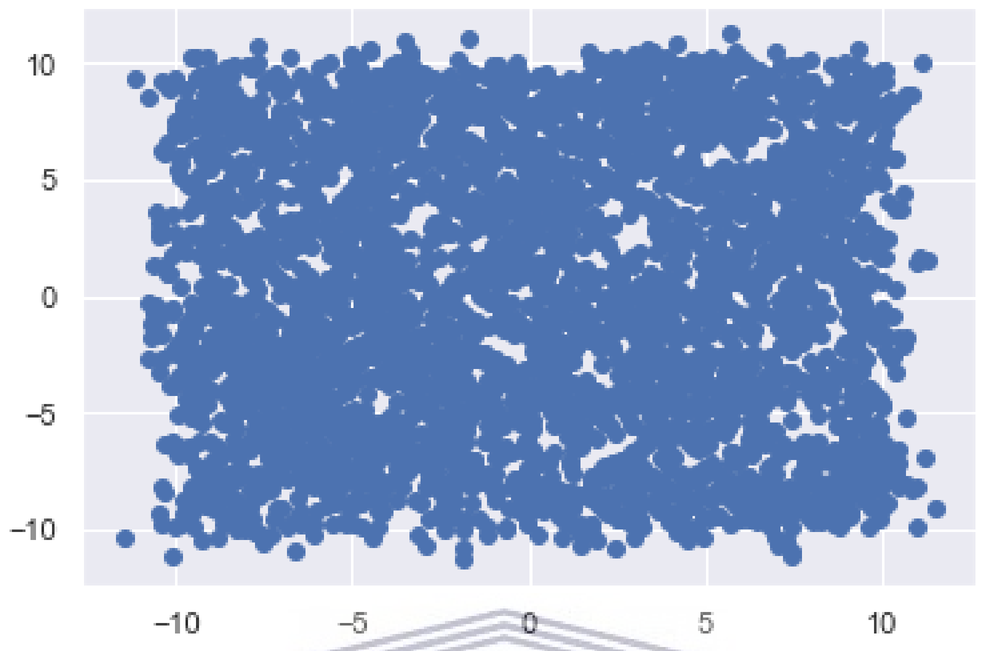


Figure 4.7: Attack traffic flow similarity grouping

```
distances = np.sort(distances, axis=0)
distances = distances[:,1]
plt.plot(distances)
```

4.2.9 Step 9 | ATDR - Discover optimal eps

DBSCAN is a base algorithm for density-based clustering containing large amounts of data which has noise and outliers. DBSCAN has two parameters - eps and minPts. Determination of the optimal eps value is acquired using the DMDBSCAN algorithm on a single density level corresponding to the k-distplot.

Figures 4.8 and 4.9 show elbow graphs obtained from running the DMBSCAN algorithm in Figure 4.10 to determine the optimal eps values for threat and legitimate traffic respectively. For threat traffic, a value of about 0.3 was obtained after close to 3800 iterations. Similar value was also obtained for the legitimate traffic but after much fewer iterations (about 275).

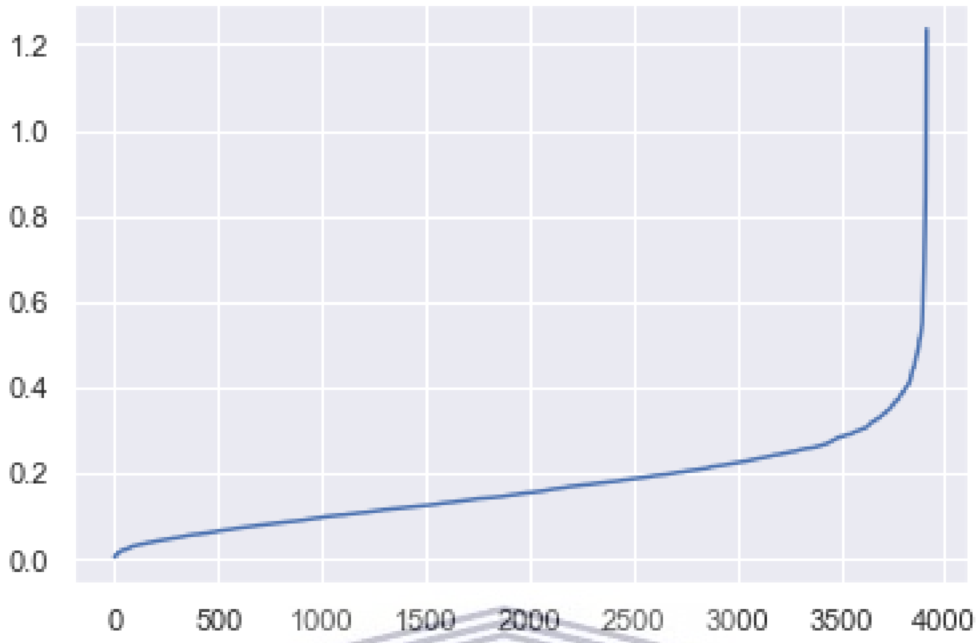


Figure 4.8: ATDR - Determine threat traffic optimal eps value required for clustering with DB-SCAN

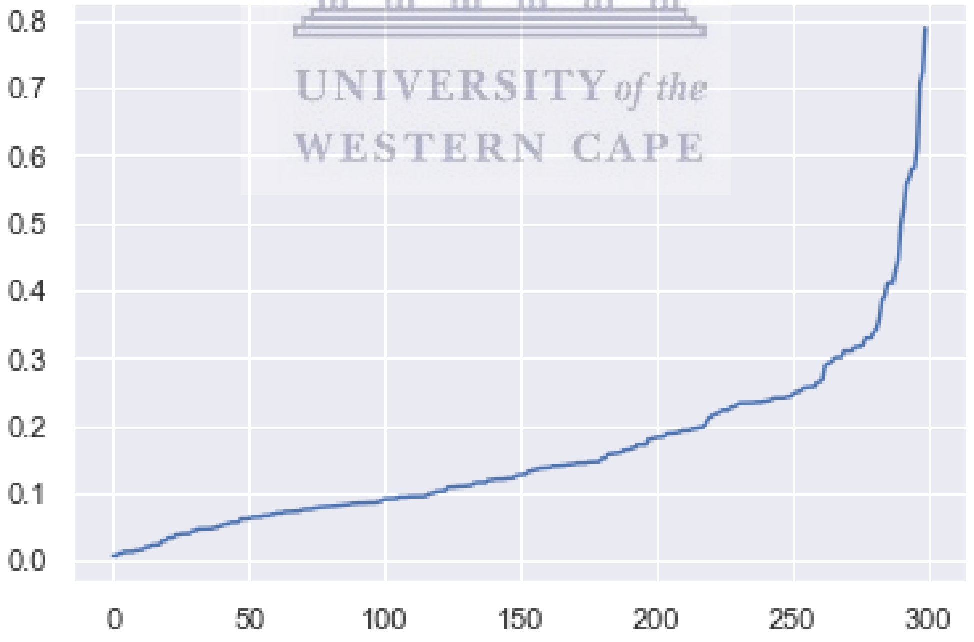
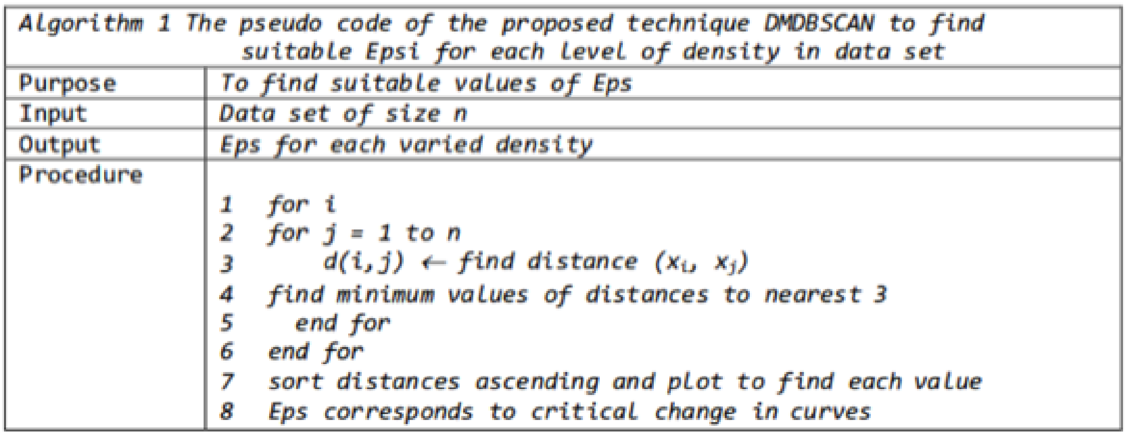


Figure 4.9: ATDR - Determine legitimate traffic optimal eps value required for clustering with DBSCAN



Algorithm 1 The pseudo code of the proposed technique DMDBSCAN to find suitable Epsi for each level of density in data set

Purpose	To find suitable values of Eps
Input	Data set of size n
Output	Eps for each varied density
Procedure	

```
for i
for j = 1 to n
    d(i,j) ← find distance ($x_i$, $x_j$)
find minimum values of distances to nearest 3
  end for
end for
sort distances ascending and plot to find each value
Eps corresponds to critical change in curves
```

Figure 4.10: DMDBSCAN algorithm (Elbatta 2012, [1])

4.2.10 Step 10 | ATDR - Clusters plot

In this final step, the obtained clusters are shown using a scatter plot.

Figure 4.11 shows four clusters from the legitimate traffic, while Figure 4.12 shows the clusters with colour coded traffic types. The Figure 4.12 also presents eight other traffic types that were identified by the model.

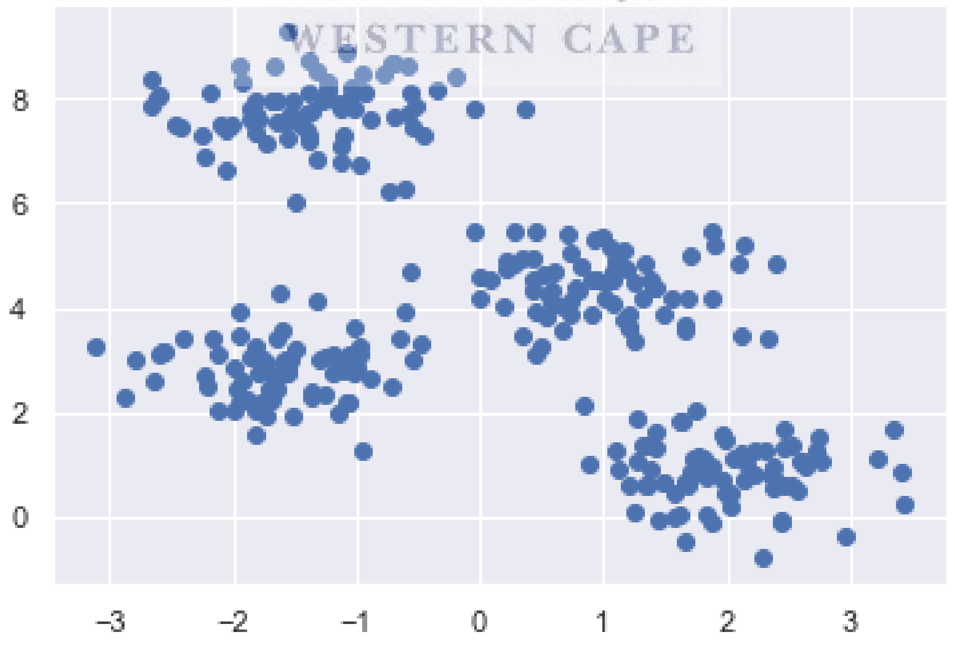


Figure 4.11: Traffic recognition formation phase 1 - Learning and grouping

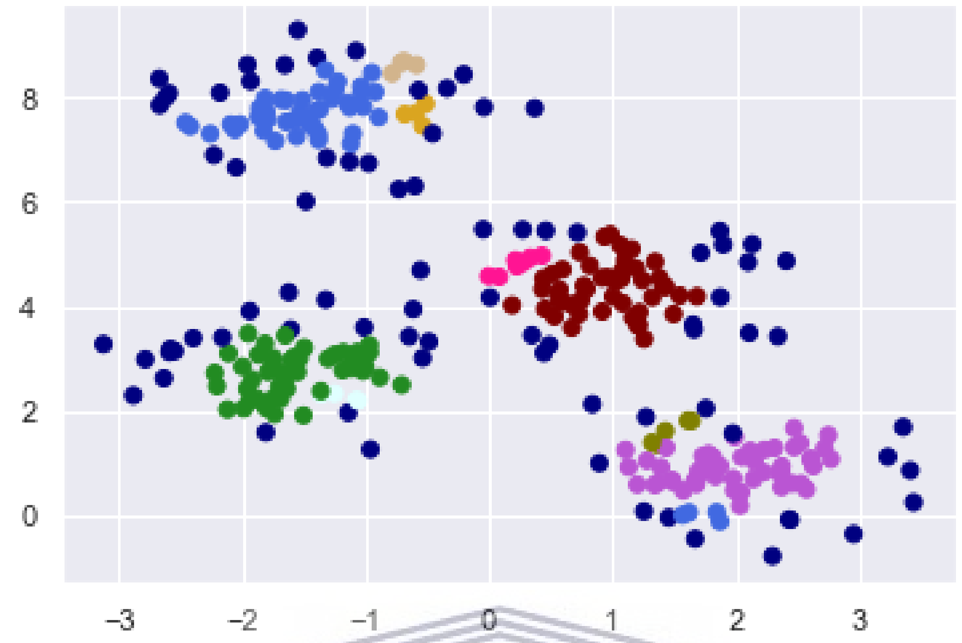


Figure 4.12: Traffic recognition formation phase 2 - Managed state

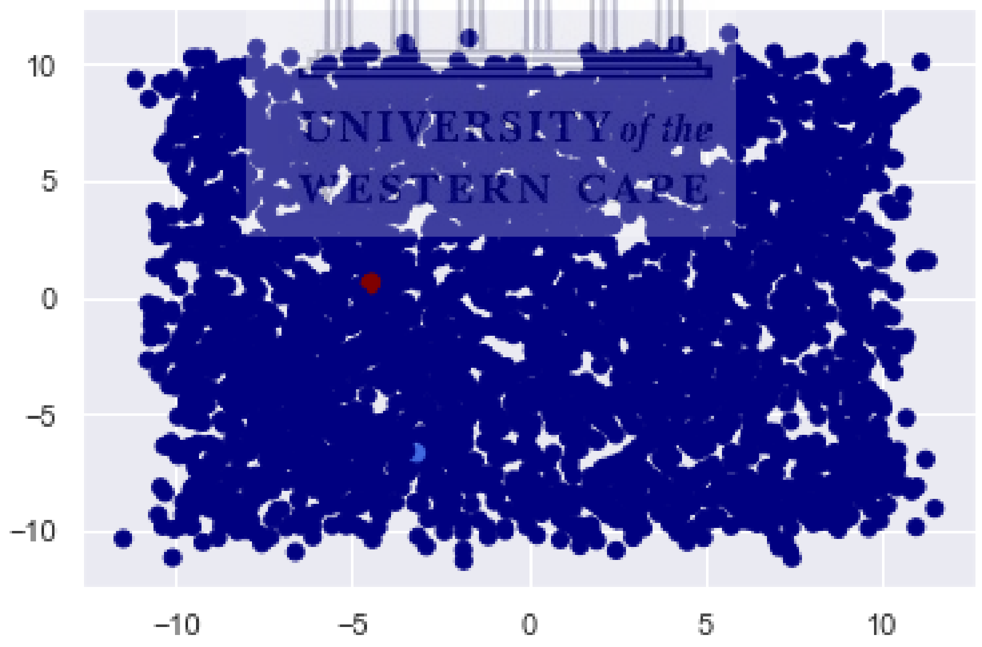


Figure 4.13: Multi-vector threat detected traffic patterns

```
# cluster the data into the ATDR_ clusters (count of grouped features)
```

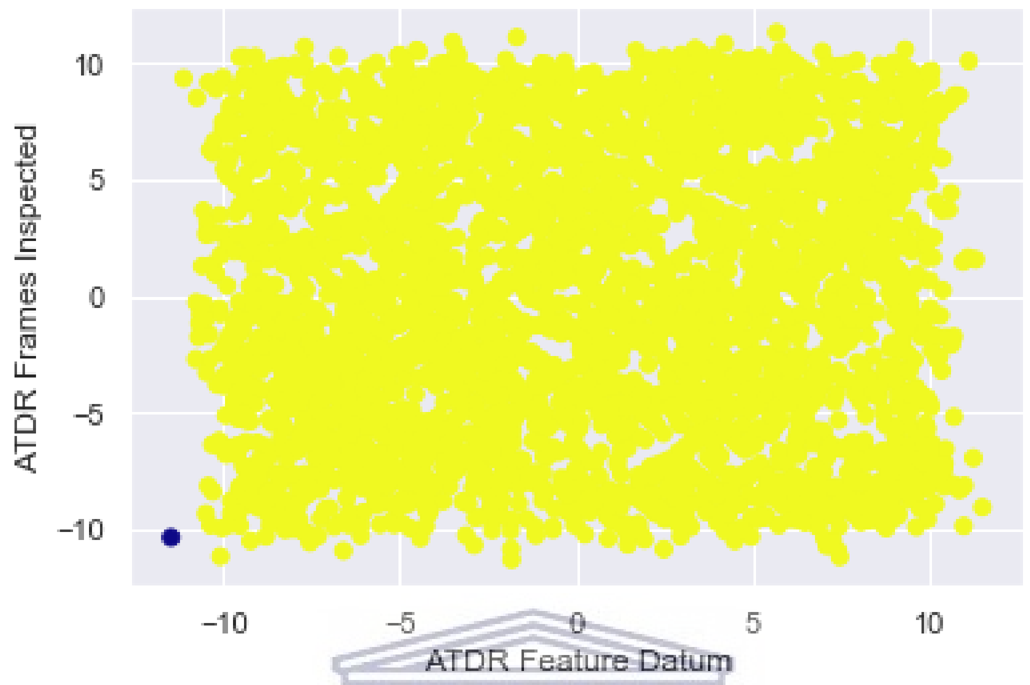


Figure 4.14: Feature datum of multi-vector threat detected traffic pattern in isolation

Figures 4.13 and Figure 4.14 respectively show the multi-vector threat detection pattern and the feature datum in isolation.

4.3 Discussion of clustering results

The results obtained thus far have show that ATDR is indeed able to categorise network traffic into groups (clusters) and is also able to detect malicious traffic patterns.

The detection (and isolation) of malicious/anomalous traffic group is more obvious when a single feature from the traffic data set is considered. Figure 4.15 and Figure 4.16 illustrates the orange cluster indicative of malicious traffic. Once identified, preventive and/or mitigation actions can then be taken.

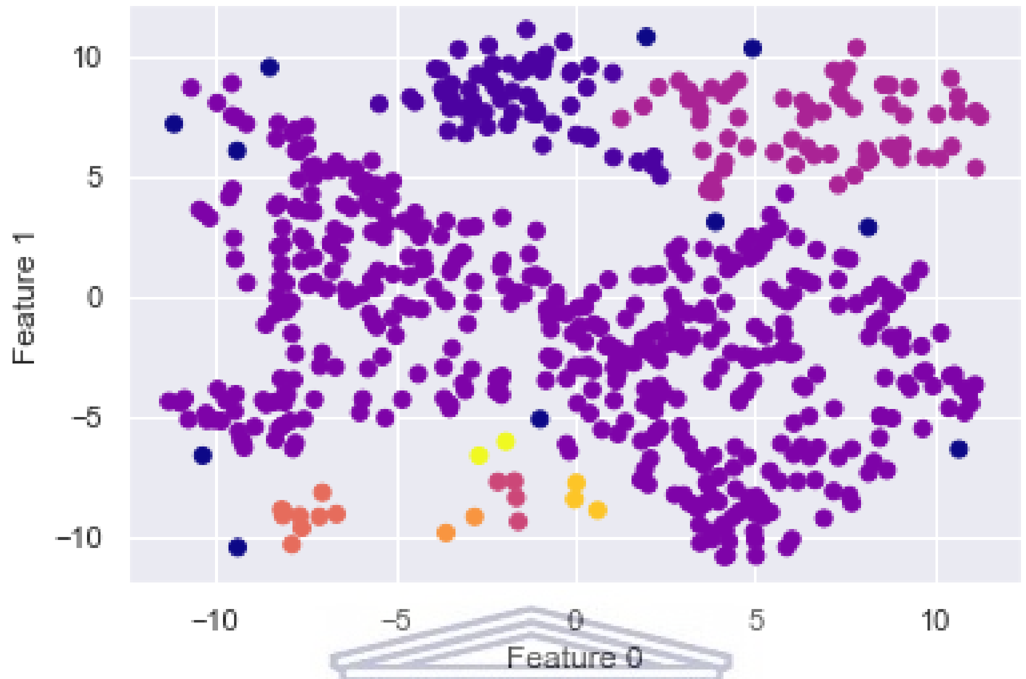


Figure 4.15: ATDR clusters forming granular groups

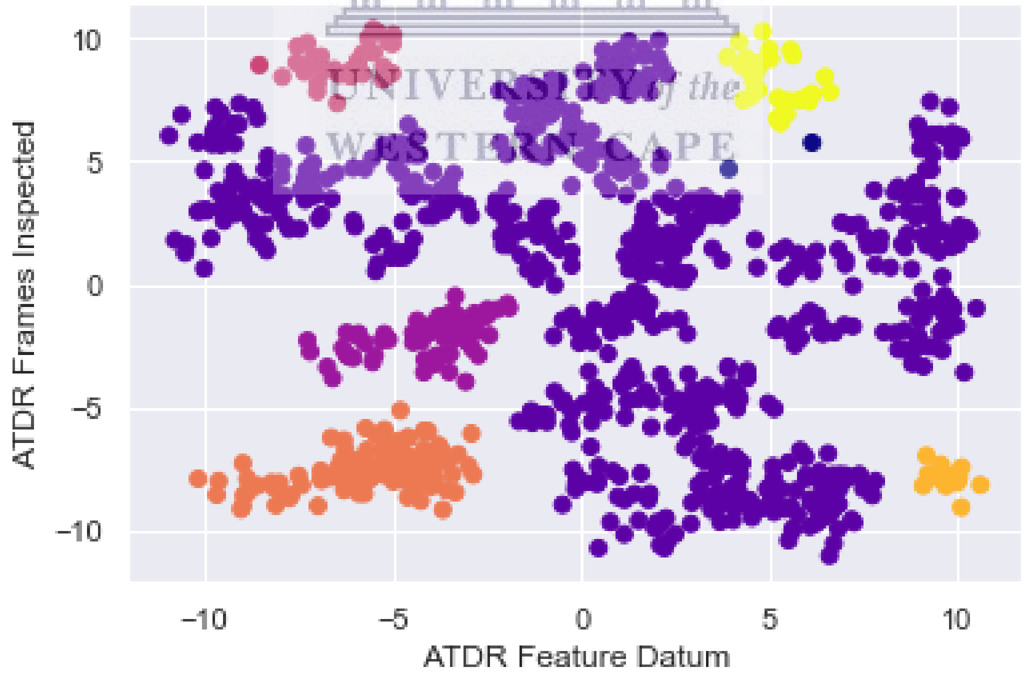


Figure 4.16: ATDR clusters fully converged and anomalous traffic grouped in the isolated orange cluster

4.4 M/M/1/N server queue processing

The dimensioning test aims to apply Erlang C function which is a well-known queuing formula with efficient computational implementations. According to [23], *ErlangC()* is a function that computes the probability that on arrival of a packet to an M/M/c system, all queues in the system are either full or it has to wait in a queue for an available server. This formula will be applied to predict the probability of packet being dropped to achieve optimal queuing parameters, similar to the prediction of probability that a traffic will have to wait in a queue due to a full system.

4.4.1 System model

The pseudo-random network simulator transmit packets toward the processing server from the test input. Varied input data are fed into the system to test the ATDR's robustness, including the discovery of the optimal scheduling time slice for optimal server processing. The network traffic flows that are steered to the server are inspected to retrieve the tuple elements of an IP packet for comparison matching using:

$$\begin{bmatrix} Protocol & Payload & Length & Flags & DstPort \\ PayloadRegex & String & HeaderChecks & Checksum & \\ PacketDirection & NumberofPackets & & & \\ PacketLength & PacketIAT & FlowDuration & & \end{bmatrix}$$

The goal is to dynamically learn and arrive at the optimal time quantum for optimal server processing. A code change modifies the simulator to schedule the end of the current time slice which equals the addition of the current simulation time and the remainder of the processing time (of the packet) or the allocated time slice, whichever has the smallest value. According to [24], usual network management methods should also monitor the partial traffic stream loads and not only the server load .

4.4.2 ATDR M/M/1/N queue

Listed below are the characteristics of the ATDR M/M/1/N queue:

1. Poisson arrivals (exponentially distributed inter-arrival times)

2. Inter-arrival times and service times are exponentially distributed

3. Single server

4. Unlimited queue size

5. Arrivals finding full queues will be dropped.

4.4.2.1 Queue processing derivatives

- The average number of packets being serviced in the queue at any given time is given by:

$$W = \frac{\sum_i n_i X(t_f - t_{f-1})}{t_f} = \frac{W + \sum_i n_i X(t_f - t_{f-1})}{t_f} \tag{4.1}$$

- Duration that the server occupies while processing the queue is given by:

$$W = \sum_i n_i X(t_i - t_{i-1}) \tag{4.2}$$

- The number of packets in the queue at any time and is given by:

$$N = \lambda * t \tag{4.3}$$

where λ is arrival rate.

- Queue utilisation:

$$P = \lambda / \mu \tag{4.4}$$

- Mean number of packets in the queue:

$$p/(1-p) \tag{4.5}$$

4.4.2.2 M/M/1/N round-robin queue | Fundamental time quantum election

$$\lambda \longrightarrow \textcircled{\mu} \supset\longrightarrow$$

M/M/1/N round-robin queue is an enhancement to the M/M/1 queue. The queue length distribution is given by $P_n = (1-\rho)\rho^n$. To meet the goal of learning the optimal time quantum for optimal server processing, a code change modifies the simulator to schedule the end of the current time slice.

Table 4.2: C++ simulator variables

Variable	Description
x	the time (in the future) that the current time-slice processing completes
Time	total simulation time at that point
this→tm	remaining processing"burst" time
q	time slice

In C++, $fmin$ returns the smaller of two arguments: either x or y.

$$fmin(x,y) = \frac{(x)as(x\ \ y)}{(y)as(y\ \ x)} \tag{4.6}$$

Per packet visits to the queue is constantly analysed to ensure the optimal efficiency of the time slice allocation at any given time using the equation below:

$$VisitsPerPacket = Ceilling\left(\frac{TotalPacketServiceTime}{TimeSliceDuration}\right) = Ceilling\left(\frac{B}{q}\right) \tag{4.7}$$

The packet visit count must be rounded up to an integer, as the "Total Packet Service Time" is not always necessarily a fraction of "Time Slice". Thus, the rescheduling of packets in a queue by the server for further processing in round-robin can be achieved with the following C++ code:

$$x = time + fmin(this \to tm, q) \tag{4.8}$$

$Schedule(x);$ *schedule the time-slice event*

4.4.3 Dimensioning

Dimensioning calculation is aimed at returning the probability of traffic flow being dropped in an M/M/1/N system using the recursive Erlang C approach.

In this study, we set $\lambda = 10, \mu = 12$, and used 50 000 traffic flows as the training set. We also compared the queue processing service times with the T_{max} prediction. This value ρ ranges between 0.50 – 0.99.

Parameters - Input:

load : float - average arrival rate * average service time (units are erlangs).

c : int - number of servers

Parameters - Output:

probability of arriving to a full system : float

probability of all servers being busy: float

4.4.4 ATDR - Traffic tail dropping

For optimal processing with respect to system capacity, stress testing of a single server system with traffic intensities measured at 1.0 was focused around discovering the probability that traffic will arrive at the system at full capacity, to which was less than 0.5. The statistical probability of traffic being dropped is returned with an Erlang C recursive approach, as shown in Figure 4.17. Scaling up with additional servers reduces the probability of tail drop, as shown in Figure 4.18.

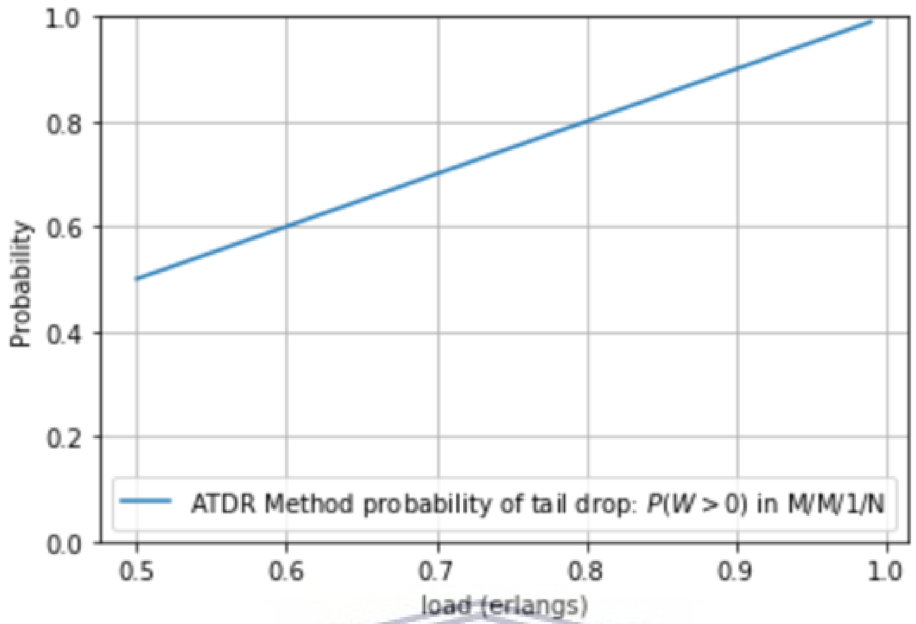


Figure 4.17: ATDR - Probability of tail dropping traffic by load

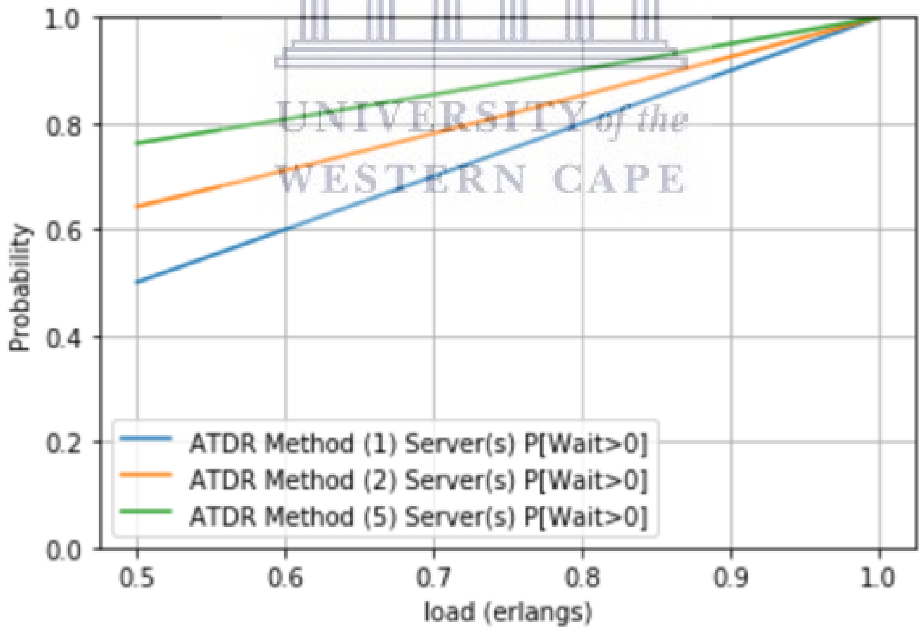


Figure 4.18: ATDR - Probability of tail dropping considering 1, 2 and 5 servers

4.4.5 Success criteria - Optimal M/M/1/N server queue processing

For traffic intensities less than 1, there is a probability that traffic would be dropped (p< 1). Optimal dimensioning is achieved by scaling to the sufficient count of servers that will mitigate the probability of any legitimate traffic discards (drops).

4.4.6 Results summary

The results of the tests between M/M/1 queue and a M/M/1/N queue with round-robin scheduling proved that round-robin scheduling enhances the completion success rate and optimises the processing time of all customers in the queue.

With a large queue size and a large time slice the total execution vastly increases which causes delay.
A large time slice and small queue causes fewer customers to be processed and a greater customer drop rate.
Customer drop rate increases when both the queue size and time slice are too small.
Discovering the optimal smaller time slice and longer queue length combination in real time yielded greater optimal server processing pertaining to total execution time and completions rates.

The results of simulation experiments are tabled in Table 4.3 and Table 4.4.

Table 4.3: Simulation queue test part 1 - Optimal queue time slice discovery

Parameters	Lambda	Mu	Time slice	Q-Length	RunTime	Avg Q-Length
M/M/1/N Round-Robin vs M/M/1 Q	10	12			4987.61	4.68759
Large TimeSlice (q);Q-Length(N)	10	12	200	1000	4987.61	4.68759
	10	12	200	1000	4987.61	4.68759
Large TimeSlice(q);Small Q-Length(N)	10	12	200	1	5772.34	0.873077
	10	12	200	2	5482.51	1.27043
	10	12	200	5	5162.98	2.29483
	10	12	200	10	5023.04	3.52317
	10	12	200	20	4990.48	4.42346
	10	12	200	40	4967.16	5.1141
Small TimeSlice(q);Large Q-Length(N)	10	12	1	200	4987.61	4.68759
	10	12	0.8	200	4987.18	4.68897

	10	12	0.5	200	4979.64	4.7497
	10	12	0.2	200	4733.56	5.06072
	10	12	0.1	200	4060.23	5.66797
	10	12	0.08	200	3762.1	5.13801
	10	12	0.05	200	3111.09	4.67976
	10	12	0.01	200	1021.92	4.38593
	10	12	0.008	200	841.899	4.1134
	10	12	0.005	200	543.662	4.71814
	10	12	0.002	200	227.519	5.77943
	10	12	0.001	200	121.387	2.97649
	10	12	0.00001	200	1.93816	0.994522
Small TimeSlice(q);Small Q-Length(N)	10	12	0.001	40	121.387	2.97649
	10	12	0.001	20	121.387	2.97649
	10	12	0.001	10	121.535	3.50923
	10	12	0.001	8	119.273	2.80713
	10	12	0.001	5	126.637	2.20504
	10	12	0.001	2	143.166	1.30702
…..	10	12	0.001	1	164.982	0.854309

Table 4.4: Simulation queue test part 2 - Optimal queue time slice discovery

Parameters	Arrivals	Completions	M/M/1 Queue length	Customers in Queue	Actors in Calendar	Traffic Dropped
M/M/1/N Round-Robin vs M/M/1 Q	50000	50000	5	0	1	
Large time slice (q); Queue length(N)	50000	50000	5	0	1	
	50000	50000	5	0	1	
Large time slice(q);Small Q-Length(N)	57812	42188	5	0	1	15624
	55045	44955	5	2	2	10087
	52005	47995	5	0	1	4010
	50681	49319	5	0	1	1362
	50092	49908	5	4	2	179
	50005	49995	5	4	2	5
Small time slice(q);Large Q-Length(N)	50000	50000	5	0	1	0
	49999	49998	5	0	2	0
	49938	49934	5	3	2	0
	47678	47668	5	9	2	0
	41189	41186	5	2	2	0
	38226	38225	5	0	2	0
	31317	31307	5	9	2	0
	10357	10355	5	1	2	0
	8566	8565	5	0	2	0
	5578	5577	5	0	2	0
	2304	2290	5	13	2	0
	1182	1181	5	0	2	0
	16	11	5	4	2	0
Small time slice(q);Small Q-Length(N)	1182	1181	5	0	2	0
	1182	1181	5	0	2	0
	1199	1168	5	0	2	30
	1193	1145	5	5	2	42
	1239	1166	5	2	2	70
	1418	1159	5	0	2	258
	1592	1191	5	0	2	400

4.5 Round-robin rescheduling switching overhead consideration

The context switch overhead associated with re-scheduling can be defined as:

Rescheduling overhead = $C/(q+C)$

where q is the length of the time slice and C is the queue rescheduling time.

An increment in q results in an increase in efficiency and a reduction in average response time. Suppose the number of arrivals is (10) , q = 200 milliseconds, and C = 5 milliseconds: Customer 0 (first in the queue of the list of Customers that are ready to run) gets to run immediately. Customer 1 can only run after customer 0's quantum expires (200 milliseconds) and the context switch takes place (5 milliseconds), so Customer 1 starts to run at 205 milliseconds. Likewise, Customer 2 can only run after another 205 milliseconds. The amount of time delay experienced by each customer can be calculated and a comparison done between delay using a small quantum (20 milliseconds) and a long quantum (200 milliseconds) as can be seen in Table 4.5 below.

Table 4.5: M/M/1/N - Queue rescheduling switching overhead consideration

Traffic Flow	Q = 20 delay (ms)	Q = 200 delay (ms)
0	0	0
1	25	205
2	50	410
3	75	615
4	100	820
5	125	1025
6	150	1230
7	175	1435
8	200	1640
9	225	1845

With ten customers and a quantum of 200 milliseconds, the last customer in the queue will have to wait nearly 2 seconds before getting another chance. This performance will not suffice for real-time processing and protection against jitter and delay for sensitive real time protocols. When the quantum is reduced to 20 milliseconds, the last customer has to wait less than a 1/4 second before it gets processed. However, the disadvantage of this is that with a small quantum, the rescheduling overhead at $(5/(20+5))$ is 20%. Thus, about a fifth of the queue processing time is consumed for rescheduling. With a quantum of 200 milliseconds, the context switching overhead is reduced to around 2.5%.

Chapter 5

CONCLUSION AND FUTURE WORK

5.1 Conclusion

The popularity of the Internet has resulted in an exponential increase in the number of interconnected and Internet-connected devices. Today, there are billions of devices sending data to or receiving data from the Internet. These include computers, laptops, mobile phones, tablets and the plethora of smart devices, such as smart homes, smart watches, connected vehicles, autonomous machines and robots, etc. Many of these devices form critical components of bigger systems (such as smart healthcare systems and intelligent transportation systems) and a failure of one of these devices or a compromise of input or output data can be catastrophic with cascaded effects. Data compromise can emanate from transmission errors, malicious attacks (including virus and worms), cyber-hackers, etc. It is therefore pertinent to concert measures to avoid such compromises.

In the recent past, numerous efforts have been proposed and implemented to checkmate data compromise and safeguard the integrity and accuracy of data sent over the Internet. Some notable efforts include: data encryption, intrusion detection and/or prevention, firewalls, and anti-viruses. These efforts, though arguably effective, require a lot of manual human intervention. Such interventions could be in terms of studying network traffic to identify and flag potentially malicious data source(s), manually populating a whitelist of safe traffic and/or a blacklist of suspicious or malicious traffic, feature extraction, optimisation, data set annotation, etc. These process are not only taxing, but highly prone to errors and omissions due to human limitations, and extremely slow. The ability to automate these functions/processes and reduce dependency on human experts will lead to more complete, accurate and task-efficient systems.

In this study, a model for automatic classification of network traffic for the purpose of detecting and responding to detection is developed. This model, tagged Autonomous Threat Detection and Response (ATDR), uses a combination of queuing theory, round-robin scheduling and machine learning (clustering) to achieve the primary objective of guaranteeing network traffic safety. Machine learning models are synonymous with the ability to annotate data sets, optimise feature sets and optimally classify, and have been extensively used for such tasks in IP traffic classification and anomaly detection.

In testing the proposed ATDR model, simulations were carried out to test the ability to appropriately group traffic types and automatically detect malicious traffic in real time. Live network traffic was first observed over a period of time to "learn" traffic patterns. Using clustering machine learning algorithms, the network traffic was then split into distinct traffic groups. Each group corresponded to a network traffic queue and was associated with a priority level. The resulting system of queues was then modeled as an M/M/1/N system, and was managed using round-robin scheduling.

The model was also tested using a variety of queue variables to determine the optimal server processing time (quantum). Obtained results show that short a quantum is desirable, as it results in the least amount of traffic delay and avails the opportunity to better separate anomalous traffic from normal traffic. However, this introduces an overhead cost of frequent rescheduling.

In conclusion, systems that leverage on machine learning (such as ATDR proposed in this study) can contribute immensely toward enhancing self-protected, next-generation networks that are resilient against the fast-growing and highly adaptive advanced threat landscape.

5.2 Future work

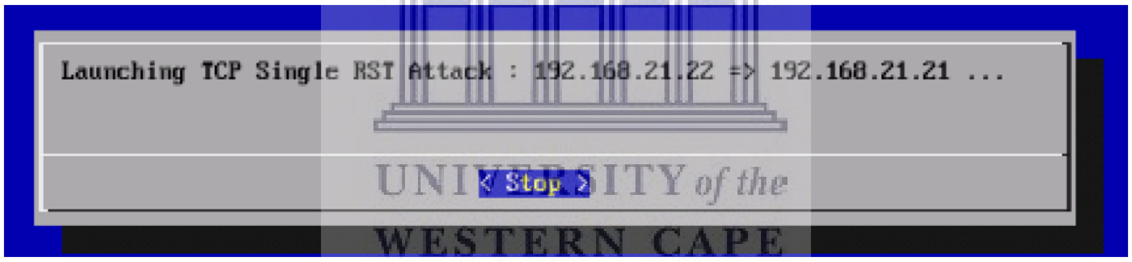
Though this study has developed a threat detection system for Internet packets, only clustering algorithms were considered. In future, it would be valuable to explore the possibilities of using deep learning models such as CNN in anomaly detection. Furthermore, only detection of malicious traffic was considered in this study. Another interesting possible future research direction could be in threat prediction. By studying and learning from historic network patterns, a machine learning model might be able to predict the likely impact of a malicious attack and curb such an attack before it takes place. Finally, extensive work on security-related challenges in self-protected networks might be another aspect of potential research.

Chapter 6

Appendix A

6.0.1 Attack simulation : TCP single RST attack simulation

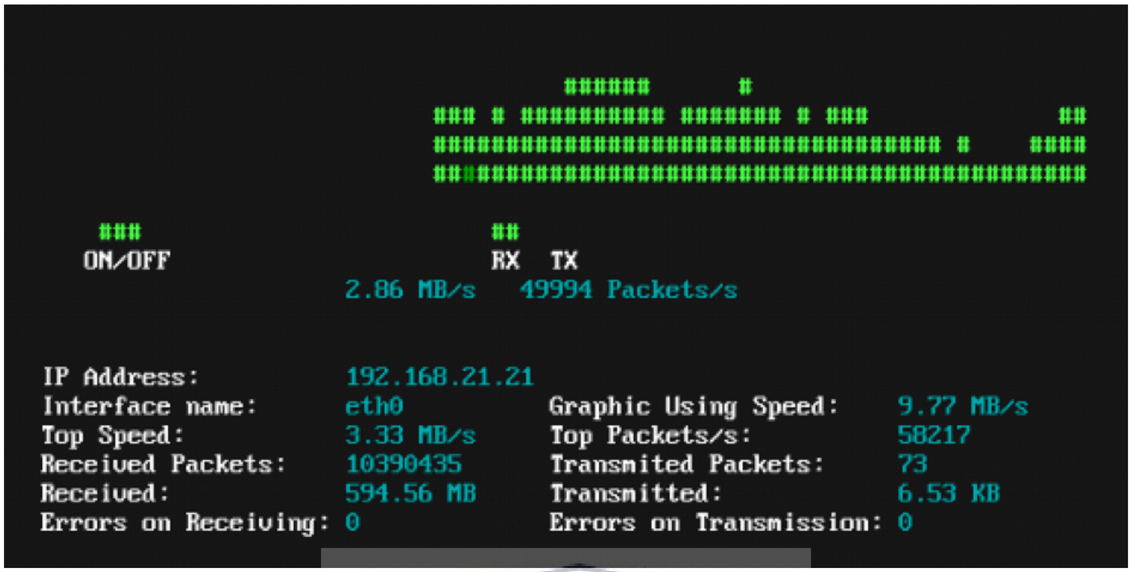
Attack launched



```
Launching TCP Single RST Attack : 192.168.21.22 => 192.168.21.21 ...
```

Figure 6.1: TCP single RST attack simulation - Launched

Victim statistics

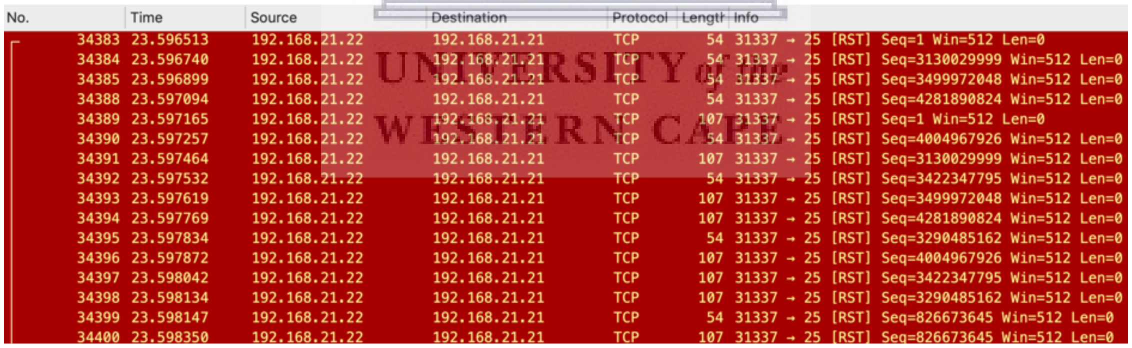


```
     ON/OFF                        RX   TX
                          2.86 MB/s   49994 Packets/s

   IP Address:            192.168.21.21
   Interface name:        eth0          Graphic Using Speed:     9.77 MB/s
   Top Speed:             3.33 MB/s     Top Packets/s:           58217
   Received Packets:      10390435      Transmited Packets:      73
   Received:              594.56 MB     Transmitted:             6.53 KB
   Errors on Receiving:   0             Errors on Transmission:  0
```

Figure 6.2: TCP single RST attack simulation - Victim statistics

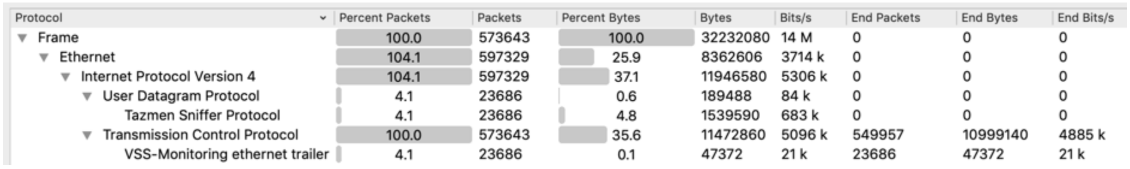
Traffic flow capture



```
No.     Time       Source          Destination     Protocol Length Info
34383  23.596513  192.168.21.22   192.168.21.21   TCP       54  31337 → 25 [RST] Seq=1 Win=512 Len=0
34384  23.596740  192.168.21.22   192.168.21.21   TCP       54  31337 → 25 [RST] Seq=3130029999 Win=512 Len=0
34385  23.596899  192.168.21.22   192.168.21.21   TCP       54  31337 → 25 [RST] Seq=3499972048 Win=512 Len=0
34388  23.597094  192.168.21.22   192.168.21.21   TCP       54  31337 → 25 [RST] Seq=4281890824 Win=512 Len=0
34389  23.597165  192.168.21.22   192.168.21.21   TCP      107  31337 → 25 [RST] Seq=1 Win=512 Len=0
34390  23.597257  192.168.21.22   192.168.21.21   TCP       54  31337 → 25 [RST] Seq=4004967926 Win=512 Len=0
34391  23.597464  192.168.21.22   192.168.21.21   TCP      107  31337 → 25 [RST] Seq=3130029999 Win=512 Len=0
34392  23.597532  192.168.21.22   192.168.21.21   TCP       54  31337 → 25 [RST] Seq=3422347795 Win=512 Len=0
34393  23.597619  192.168.21.22   192.168.21.21   TCP      107  31337 → 25 [RST] Seq=3499972048 Win=512 Len=0
34394  23.597769  192.168.21.22   192.168.21.21   TCP      107  31337 → 25 [RST] Seq=4281890824 Win=512 Len=0
34395  23.597834  192.168.21.22   192.168.21.21   TCP       54  31337 → 25 [RST] Seq=3290485162 Win=512 Len=0
34396  23.597872  192.168.21.22   192.168.21.21   TCP      107  31337 → 25 [RST] Seq=4004967926 Win=512 Len=0
34397  23.598042  192.168.21.22   192.168.21.21   TCP      107  31337 → 25 [RST] Seq=3422347795 Win=512 Len=0
34398  23.598134  192.168.21.22   192.168.21.21   TCP      107  31337 → 25 [RST] Seq=3290485162 Win=512 Len=0
34399  23.598147  192.168.21.22   192.168.21.21   TCP       54  31337 → 25 [RST] Seq=826673645 Win=512 Len=0
34400  23.598350  192.168.21.22   192.168.21.21   TCP      107  31337 → 25 [RST] Seq=826673645 Win=512 Len=0
```

Figure 6.3: ATDR method schematic

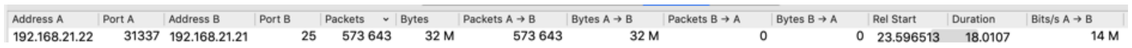
Protocol hierarchy



Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets	End Bytes	End Bits/s
▼ Frame	100.0	573643	100.0	32232080	14 M	0	0	0
▼ Ethernet	104.1	597329	25.9	8362606	3714 k	0	0	0
▼ Internet Protocol Version 4	104.1	597329	37.1	11946580	5306 k	0	0	0
▼ User Datagram Protocol	4.1	23686	0.6	189488	84 k	0	0	0
Tazmen Sniffer Protocol	4.1	23686	4.8	1539590	683 k	0	0	0
▼ Transmission Control Protocol	100.0	573643	35.6	11472860	5096 k	549957	10999140	4885 k
VSS-Monitoring ethernet trailer	4.1	23686	0.1	47372	21 k	23686	47372	21 k

Figure 6.4: TCP single RST attack simulation - Protocol hierarchy

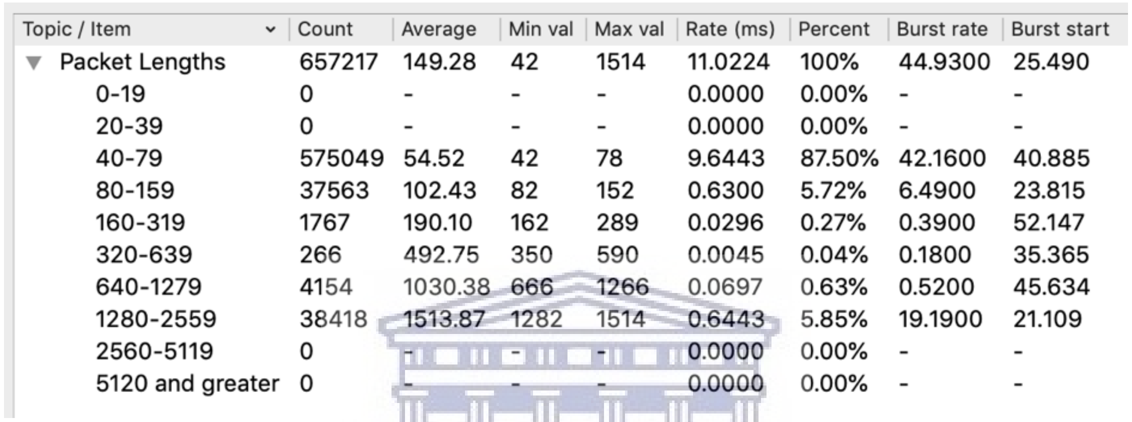
Conversation statistics



Address A	Port A	Address B	Port B	Packets	Bytes	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start	Duration	Bits/s A → B
192.168.21.22	31337	192.168.21.21	25	573 643	32 M	573 643	32 M	0	0	23.596513	18.0107	14 M

Figure 6.5: TCP single RST attack simulation - Conversations

Packet lengths



Topic / Item	Count	Average	Min val	Max val	Rate (ms)	Percent	Burst rate	Burst start
▼ Packet Lengths	657217	149.28	42	1514	11.0224	100%	44.9300	25.490
0-19	0	-	-	-	0.0000	0.00%	-	-
20-39	0	-	-	-	0.0000	0.00%	-	-
40-79	575049	54.52	42	78	9.6443	87.50%	42.1600	40.885
80-159	37563	102.43	82	152	0.6300	5.72%	6.4900	23.815
160-319	1767	190.10	162	289	0.0296	0.27%	0.3900	52.147
320-639	266	492.75	350	590	0.0045	0.04%	0.1800	35.365
640-1279	4154	1030.38	666	1266	0.0697	0.63%	0.5200	45.634
1280-2559	38418	1513.87	1282	1514	0.6443	5.85%	19.1900	21.109
2560-5119	0	-	-	-	0.0000	0.00%	-	-
5120 and greater	0	-	-	-	0.0000	0.00%	-	-

Figure 6.6: TCP single RST attack simulation - Packet lengths

Packets per second graph

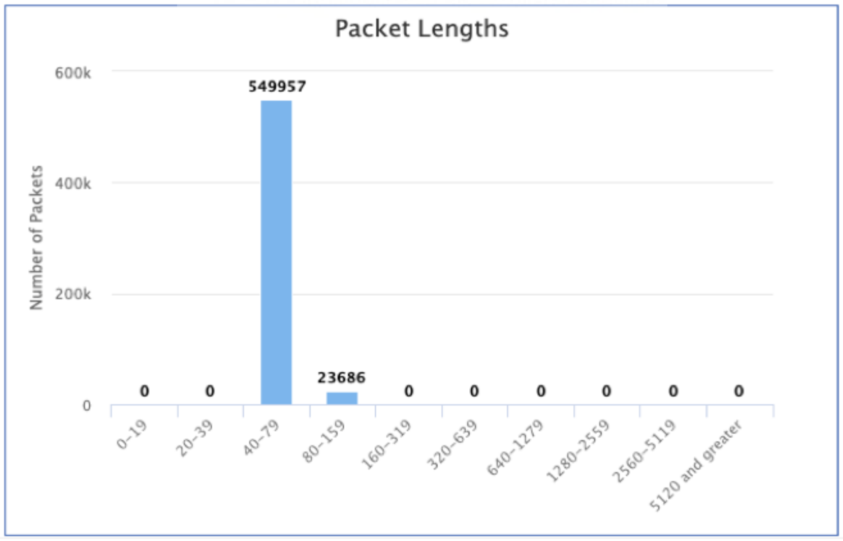


Figure 6.7: TCP single RST attack simulation - Packets per second

IPv4 statistics

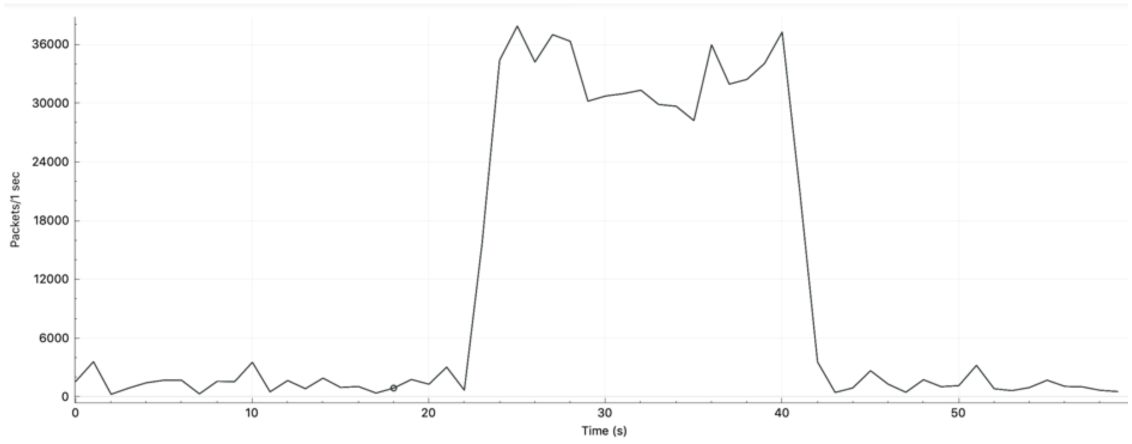
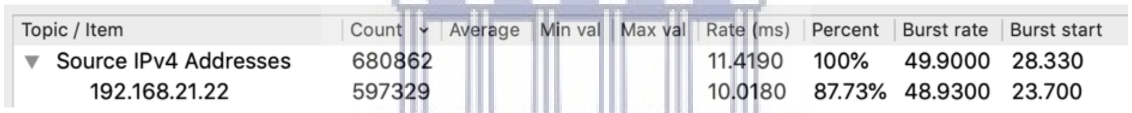


Figure 6.8: TCP single RST attack simulation - IPv4 Statistics

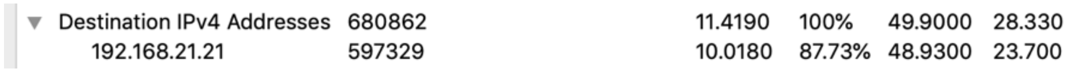
Source



Topic / Item	Count	Average	Min val	Max val	Rate (ms)	Percent	Burst rate	Burst start
▼ Source IPv4 Addresses	680862				11.4190	100%	49.9000	28.330
192.168.21.22	597329				10.0180	87.73%	48.9300	23.700

Figure 6.9: TCP single RST attack simulation - Source

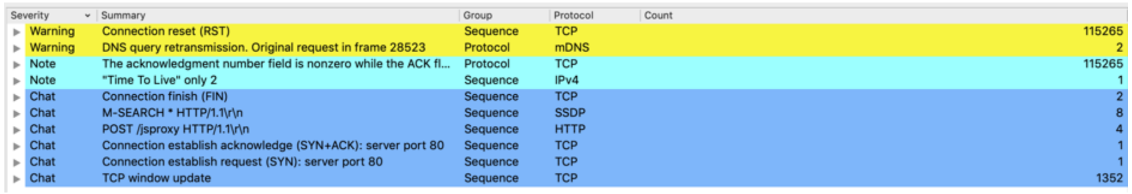
Destination



▼ Destination IPv4 Addresses	680862	11.4190	100%	49.9000	28.330
192.168.21.21	597329	10.0180	87.73%	48.9300	23.700

Figure 6.10: TCP single RST attack simulation - Destination

Expert information



Severity	Summary	Group	Protocol	Count
▶ Warning	Connection reset (RST)	Sequence	TCP	115265
▶ Warning	DNS query retransmission. Original request in frame 28523	Protocol	mDNS	2
▶ Note	The acknowledgment number field is nonzero while the ACK fl...	Protocol	TCP	115265
▶ Note	"Time To Live" only 2	Sequence	IPv4	1
▶ Chat	Connection finish (FIN)	Sequence	TCP	2
▶ Chat	M-SEARCH * HTTP/1.1\r\n	Sequence	SSDP	8
▶ Chat	POST /jsproxy HTTP/1.1\r\n	Sequence	HTTP	4
▶ Chat	Connection establish acknowledge (SYN+ACK): server port 80	Sequence	TCP	1
▶ Chat	Connection establish request (SYN): server port 80	Sequence	TCP	1
▶ Chat	TCP window update	Sequence	TCP	1352

Figure 6.11: TCP Single RST Attack Simulation - Expert information

Sequence numbers graph

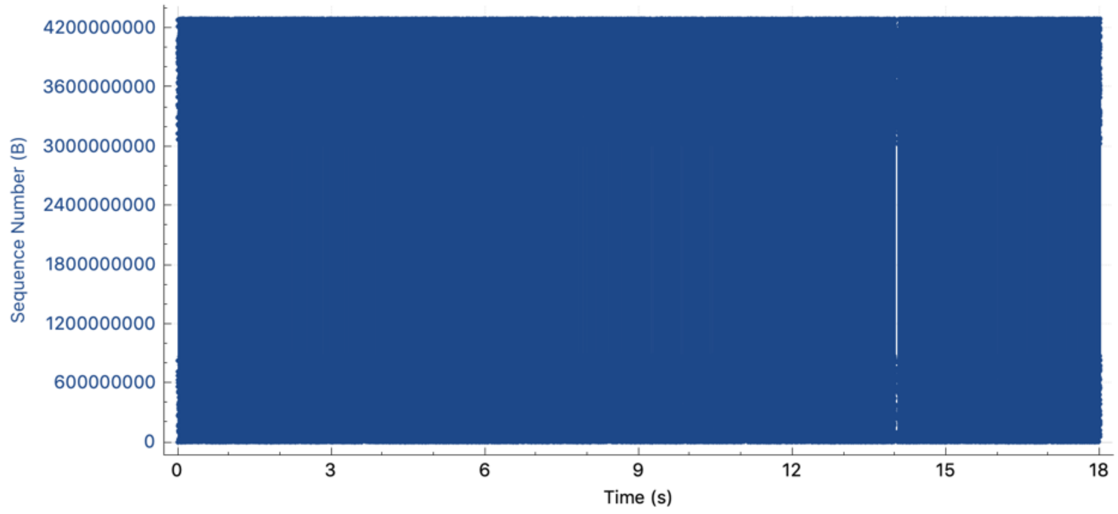
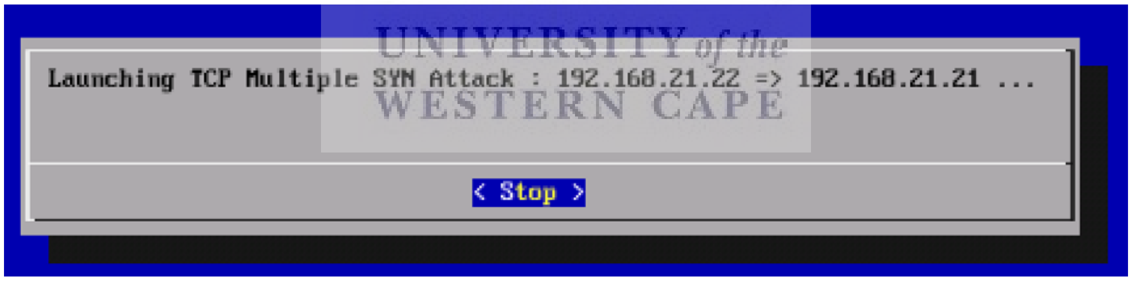


Figure 6.12: TCP single RST attack simulation - Sequence numbers

6.0.2 Attack simulation : Single SYN attack simulation

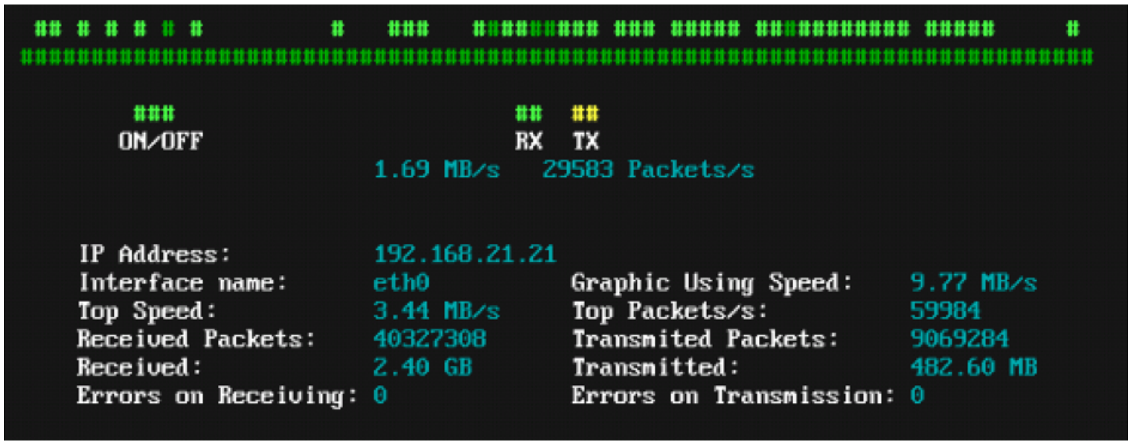
Attack launched



```
Launching TCP Multiple SYN Attack : 192.168.21.22 => 192.168.21.21 ...
```

Figure 6.13: Single SYN attacks simulation - Attack launched

Victim statistics



```
     ###                          ##  ##
    ON/OFF                        RX  TX
                        1.69 MB/s    29583 Packets/s


  IP Address:           192.168.21.21
  Interface name:       eth0           Graphic Using Speed:    9.77 MB/s
  Top Speed:            3.44 MB/s      Top Packets/s:          59984
  Received Packets:     40327308       Transmited Packets:     9069284
  Received:             2.40 GB        Transmitted:            482.60 MB
  Errors on Receiving:  0              Errors on Transmission: 0
```

Figure 6.14: Single SYN attacks simulation - Victim statistics

Packets per second Graph

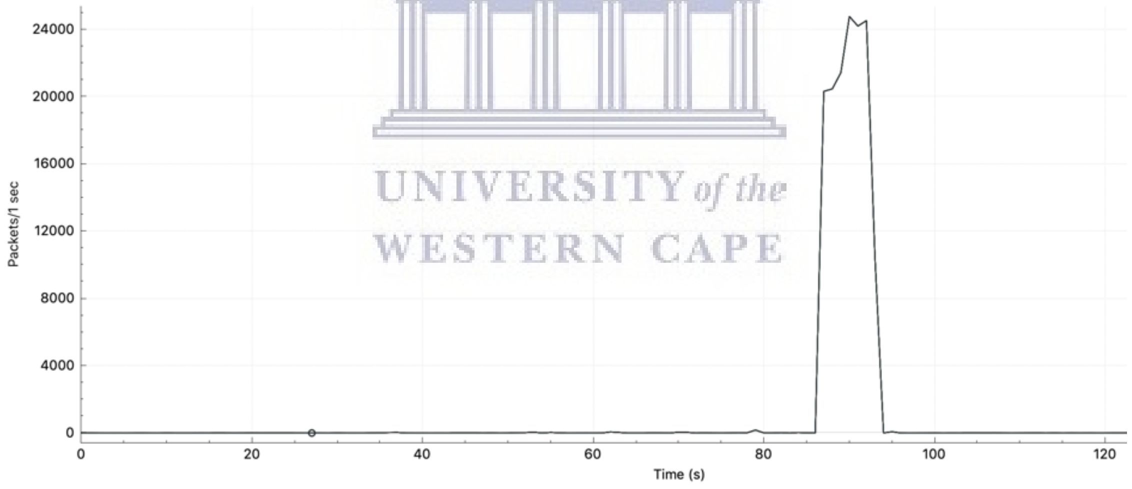
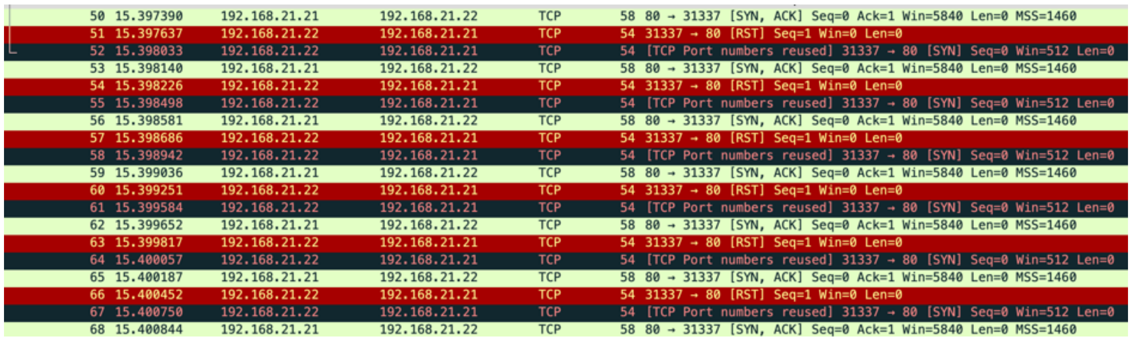


Figure 6.15: Single SYN attacks simulation - Packets per second

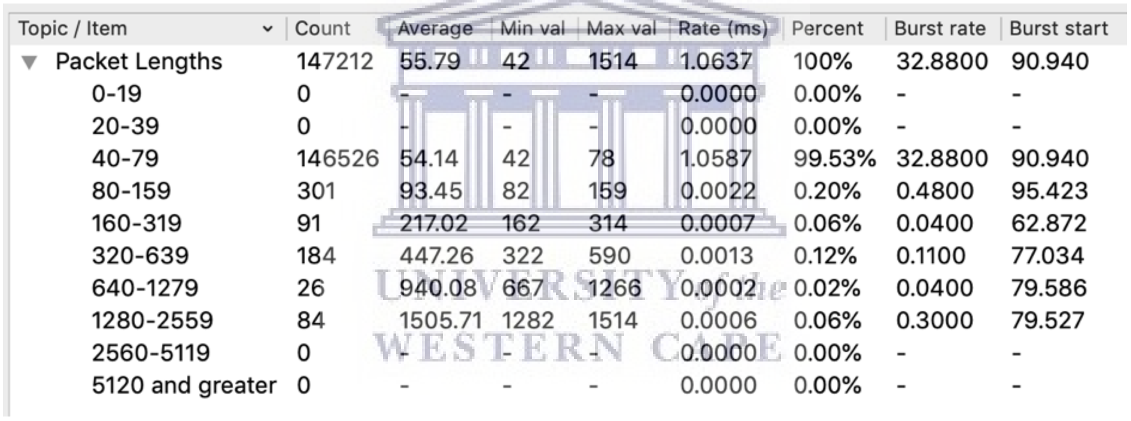
Traffic flow capture



```
50 15.397390  192.168.21.21  192.168.21.22  TCP  58 80 → 31337 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460
51 15.397637  192.168.21.22  192.168.21.21  TCP  54 31337 → 80 [RST] Seq=1 Win=0 Len=0
52 15.398033  192.168.21.22  192.168.21.21  TCP  54 [TCP Port numbers reused] 31337 → 80 [SYN] Seq=0 Win=512 Len=0
53 15.398140  192.168.21.21  192.168.21.22  TCP  58 80 → 31337 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460
54 15.398226  192.168.21.22  192.168.21.21  TCP  54 31337 → 80 [RST] Seq=1 Win=0 Len=0
55 15.398498  192.168.21.22  192.168.21.21  TCP  54 [TCP Port numbers reused] 31337 → 80 [SYN] Seq=0 Win=512 Len=0
56 15.398581  192.168.21.21  192.168.21.22  TCP  58 80 → 31337 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460
57 15.398686  192.168.21.22  192.168.21.21  TCP  54 31337 → 80 [RST] Seq=1 Win=0 Len=0
58 15.398942  192.168.21.22  192.168.21.21  TCP  54 [TCP Port numbers reused] 31337 → 80 [SYN] Seq=0 Win=512 Len=0
59 15.399036  192.168.21.21  192.168.21.22  TCP  58 80 → 31337 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460
60 15.399251  192.168.21.22  192.168.21.21  TCP  54 31337 → 80 [RST] Seq=1 Win=0 Len=0
61 15.399584  192.168.21.22  192.168.21.21  TCP  54 [TCP Port numbers reused] 31337 → 80 [SYN] Seq=0 Win=512 Len=0
62 15.399652  192.168.21.21  192.168.21.22  TCP  58 80 → 31337 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460
63 15.399817  192.168.21.22  192.168.21.21  TCP  54 31337 → 80 [RST] Seq=1 Win=0 Len=0
64 15.400057  192.168.21.22  192.168.21.21  TCP  54 [TCP Port numbers reused] 31337 → 80 [SYN] Seq=0 Win=512 Len=0
65 15.400187  192.168.21.21  192.168.21.22  TCP  58 80 → 31337 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460
66 15.400452  192.168.21.22  192.168.21.21  TCP  54 31337 → 80 [RST] Seq=1 Win=0 Len=0
67 15.400750  192.168.21.22  192.168.21.21  TCP  54 [TCP Port numbers reused] 31337 → 80 [SYN] Seq=0 Win=512 Len=0
68 15.400844  192.168.21.21  192.168.21.22  TCP  58 80 → 31337 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460
```

Figure 6.16: Single SYN attacks simulation - Traffic flow capture

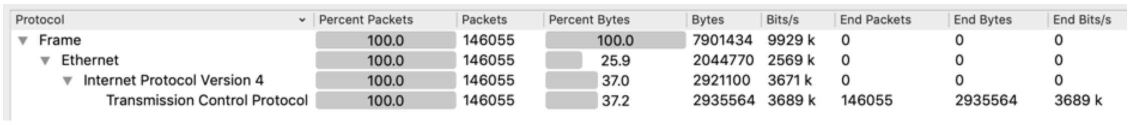
Packet lengths



Topic / Item	Count	Average	Min val	Max val	Rate (ms)	Percent	Burst rate	Burst start
▼ Packet Lengths	147212	55.79	42	1514	1.0637	100%	32.8800	90.940
0-19	0	-	-	-	0.0000	0.00%	-	-
20-39	0	-	-	-	0.0000	0.00%	-	-
40-79	146526	54.14	42	78	1.0587	99.53%	32.8800	90.940
80-159	301	93.45	82	159	0.0022	0.20%	0.4800	95.423
160-319	91	217.02	162	314	0.0007	0.06%	0.0400	62.872
320-639	184	447.26	322	590	0.0013	0.12%	0.1100	77.034
640-1279	26	940.08	667	1266	0.0002	0.02%	0.0400	79.586
1280-2559	84	1505.71	1282	1514	0.0006	0.06%	0.3000	79.527
2560-5119	0	-	-	-	0.0000	0.00%	-	-
5120 and greater	0	-	-	-	0.0000	0.00%	-	-

Figure 6.17: Single SYN Attacks Simulation - Packet lengths

Protocol hierarchy



Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets	End Bytes	End Bits/s
▼ Frame	100.0	146055	100.0	7901434	9929 k	0	0	0
▼ Ethernet	100.0	146055	25.9	2044770	2569 k	0	0	0
▼ Internet Protocol Version 4	100.0	146055	37.0	2921100	3671 k	0	0	0
Transmission Control Protocol	100.0	146055	37.2	2935564	3689 k	146055	2935564	3689 k

Figure 6.18: Single SYN attacks simulation - Protocol hierarchy

Sequence numbers graph

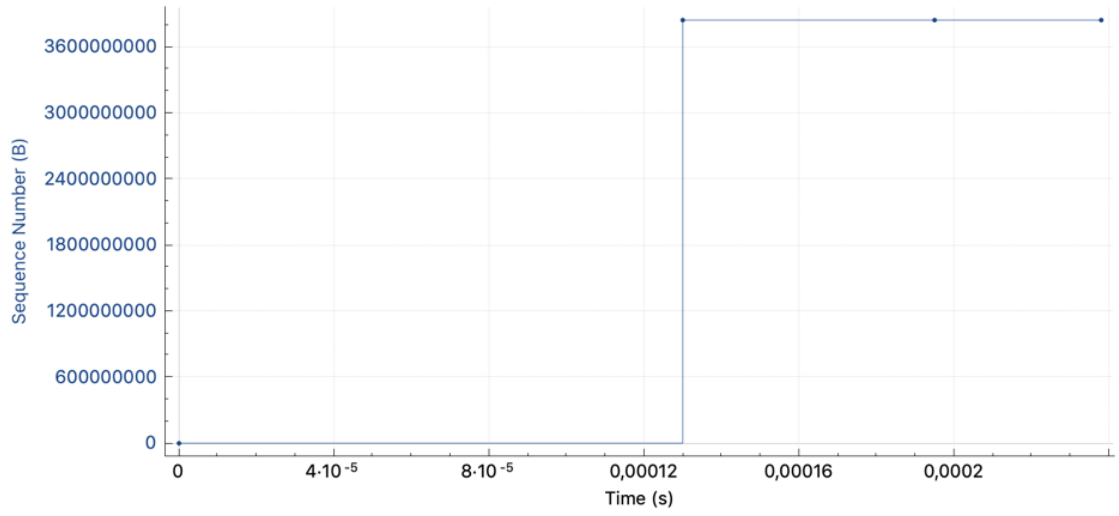
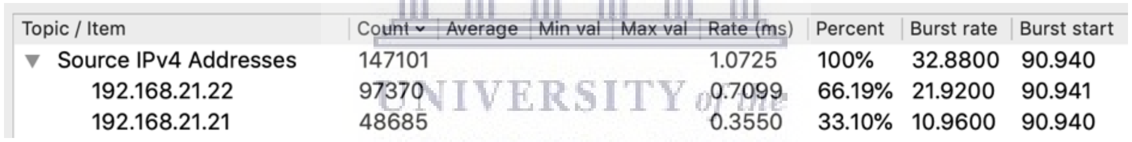


Figure 6.19: Single SYN attacks simulation - Sequence numbers

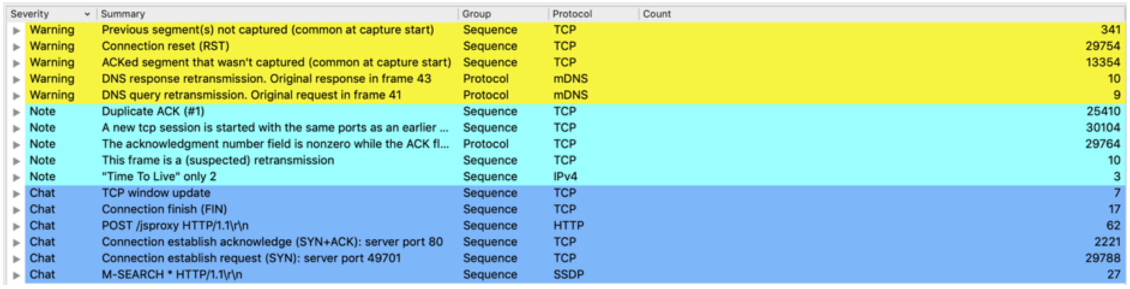
IPv4 source and destination statistics



Topic / Item	Count	Average	Min val	Max val	Rate (ms)	Percent	Burst rate	Burst start
▼ Source IPv4 Addresses	147101				1.0725	100%	32.8800	90.940
192.168.21.22	97370				0.7099	66.19%	21.9200	90.941
192.168.21.21	48685				0.3550	33.10%	10.9600	90.940

Figure 6.20: Single SYN attacks simulation - IPv4 statistics

Expert information

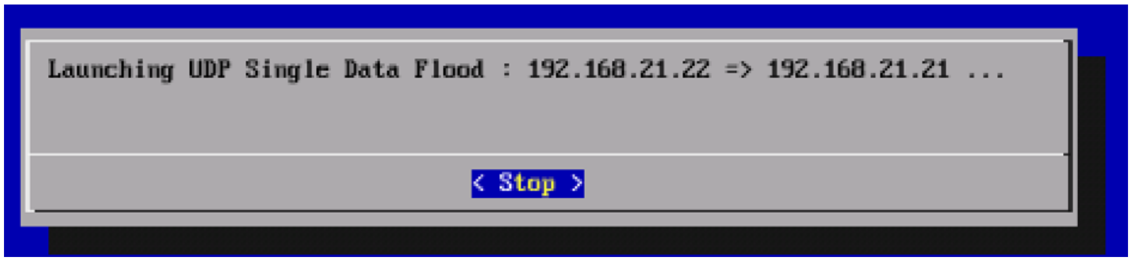


Severity	Summary	Group	Protocol	Count
Warning	Previous segment(s) not captured (common at capture start)	Sequence	TCP	341
Warning	Connection reset (RST)	Sequence	TCP	29754
Warning	ACKed segment that wasn't captured (common at capture start)	Sequence	TCP	13354
Warning	DNS response retransmission. Original response in frame 43	Protocol	mDNS	10
Warning	DNS query retransmission. Original request in frame 41	Protocol	mDNS	9
Note	Duplicate ACK (#1)	Sequence	TCP	25410
Note	A new tcp session is started with the same ports as an earlier ...	Sequence	TCP	30104
Note	The acknowledgment number field is nonzero while the ACK fl...	Protocol	TCP	29764
Note	This frame is a (suspected) retransmission	Sequence	TCP	10
Note	"Time To Live" only 2	Sequence	IPv4	3
Chat	TCP window update	Sequence	TCP	7
Chat	Connection finish (FIN)	Sequence	TCP	17
Chat	POST /jsproxy HTTP/1.1\r\n	Sequence	HTTP	62
Chat	Connection establish acknowledge (SYN+ACK): server port 80	Sequence	TCP	2221
Chat	Connection establish request (SYN): server port 49701	Sequence	TCP	29788
Chat	M-SEARCH * HTTP/1.1\r\n	Sequence	SSDP	27

Figure 6.21: Single SYN attacks simulation - Expert information

6.0.3 Attack simulation : UDP flood attack simulation

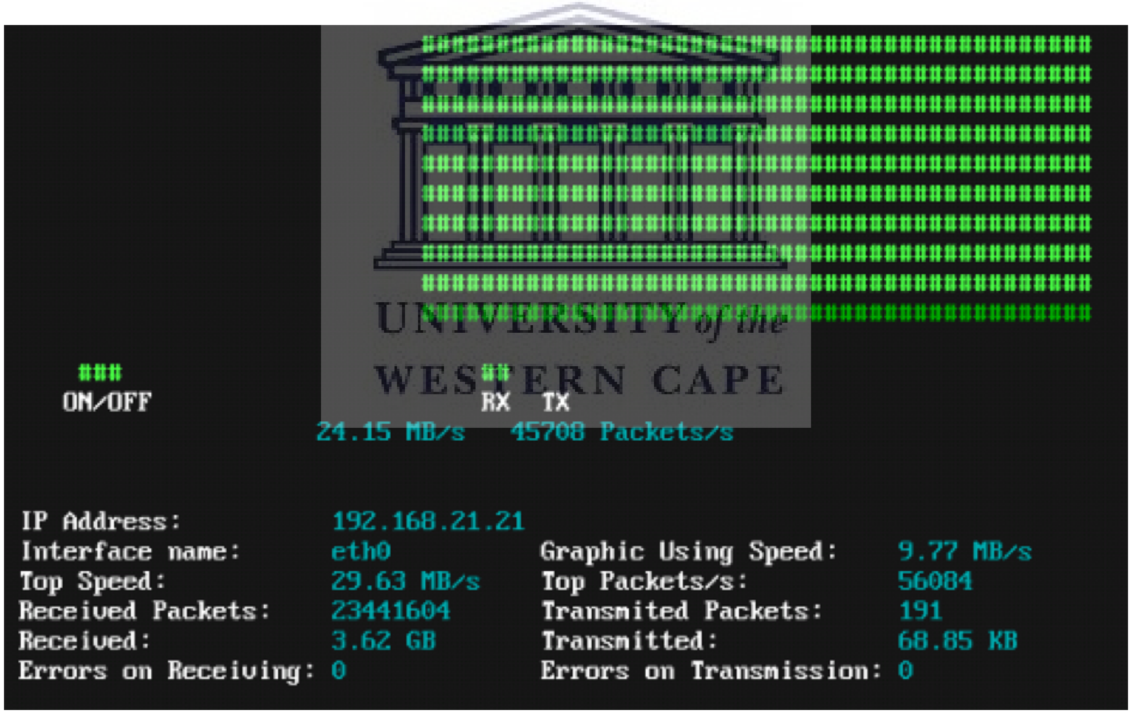
Attack started



```
Launching UDP Single Data Flood : 192.168.21.22 => 192.168.21.21 ...
```

Figure 6.22: UDP flood attack simulation - Attack launched

Victim statistics



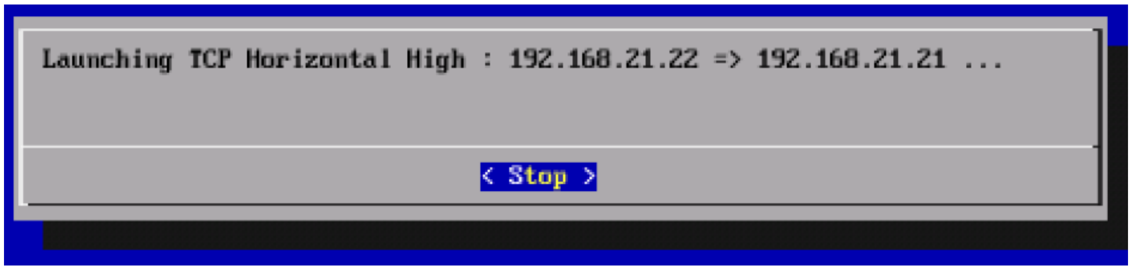
```
 ###                            ##
 ON/OFF                         RX   TX
                  24.15 MB/s   45708 Packets/s

IP Address:            192.168.21.21
Interface name:        eth0             Graphic Using Speed:      9.77 MB/s
Top Speed:             29.63 MB/s       Top Packets/s:            56084
Received Packets:      23441604         Transmited Packets:       191
Received:              3.62 GB          Transmitted:              68.85 KB
Errors on Receiving:   0                Errors on Transmission:   0
```

Figure 6.23: UDP flood attack simulation - Victim statistics

6.0.4 Attack simulation : TCP scan simulation - Type horizontal, high volume, layer 4 | using IP

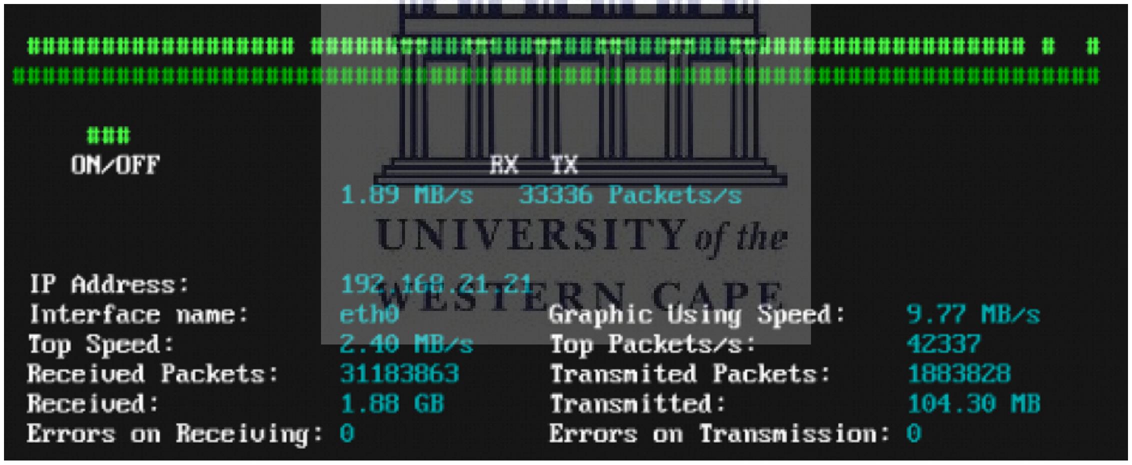
Attack launched



```
Launching TCP Horizontal High : 192.168.21.22 => 192.168.21.21 ...
```

Figure 6.24: TCP horizontal scan - Attack launched

Victim statistics

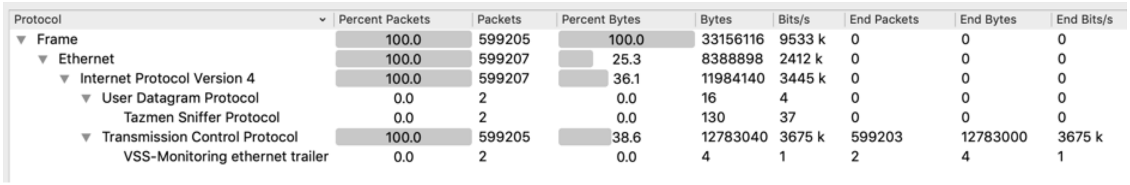


```
###
ON/OFF                         RX   TX
                     1.89 MB/s    33336 Packets/s

IP Address:          192.168.21.21
Interface name:      eth0          Graphic Using Speed:     9.77 MB/s
Top Speed:           2.40 MB/s     Top Packets/s:           42337
Received Packets:    31183863      Transmited Packets:      1883828
Received:            1.88 GB       Transmitted:             104.30 MB
Errors on Receiving: 0             Errors on Transmission:  0
```

Figure 6.25: TCP scan - Victim statistics

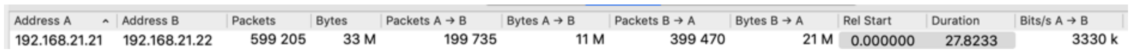
Protocol hierarchy



Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets	End Bytes	End Bits/s
▼ Frame	100.0	599205	100.0	33156116	9533 k	0	0	0
▼ Ethernet	100.0	599207	25.3	8388898	2412 k	0	0	0
▼ Internet Protocol Version 4	100.0	599207	36.1	11984140	3445 k	0	0	0
▼ User Datagram Protocol	0.0	2	0.0	16	4	0	0	0
Tazmen Sniffer Protocol	0.0	2	0.0	130	37	0	0	0
▼ Transmission Control Protocol	100.0	599205	38.6	12783040	3675 k	599203	12783000	3675 k
VSS-Monitoring ethernet trailer	0.0	2	0.0	4	1	2	4	1

Figure 6.26: TCP horizontal scan - Protocol hierarchy

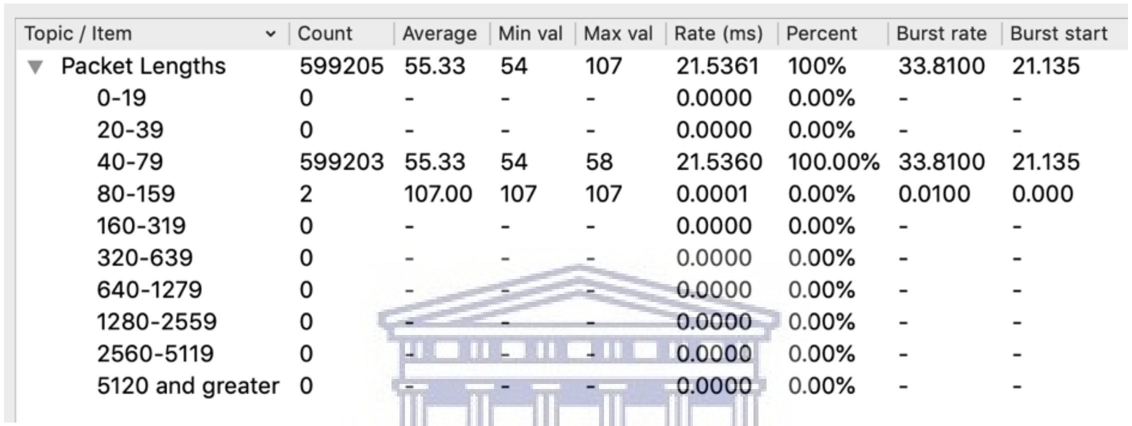
Conversations



Address A	Address B	Packets	Bytes	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start	Duration	Bits/s A → B
192.168.21.21	192.168.21.22	599 205	33 M	199 735	11 M	399 470	21 M	0.000000	27.8233	3330 k

Figure 6.27: TCP horizontal scan - Conversations

Packet Lengths



Topic / Item	Count	Average	Min val	Max val	Rate (ms)	Percent	Burst rate	Burst start
▼ Packet Lengths	599205	55.33	54	107	21.5361	100%	33.8100	21.135
0-19	0	-	-	-	0.0000	0.00%	-	-
20-39	0	-	-	-	0.0000	0.00%	-	-
40-79	599203	55.33	54	58	21.5360	100.00%	33.8100	21.135
80-159	2	107.00	107	107	0.0001	0.00%	0.0100	0.000
160-319	0	-	-	-	0.0000	0.00%	-	-
320-639	0	-	-	-	0.0000	0.00%	-	-
640-1279	0	-	-	-	0.0000	0.00%	-	-
1280-2559	0	-	-	-	0.0000	0.00%	-	-
2560-5119	0	-	-	-	0.0000	0.00%	-	-
5120 and greater	0	-	-	-	0.0000	0.00%	-	-

Figure 6.28: TCP horizontal scan - Packet lengths

Packets Per Second Graph

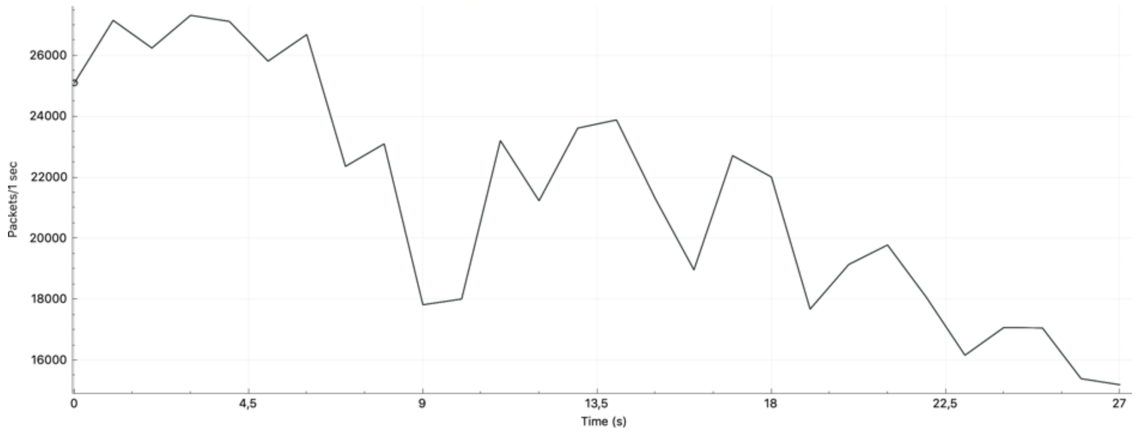


Figure 6.29: TCP Horizontal Scan - Packets per second

Sequence Numbers Graph

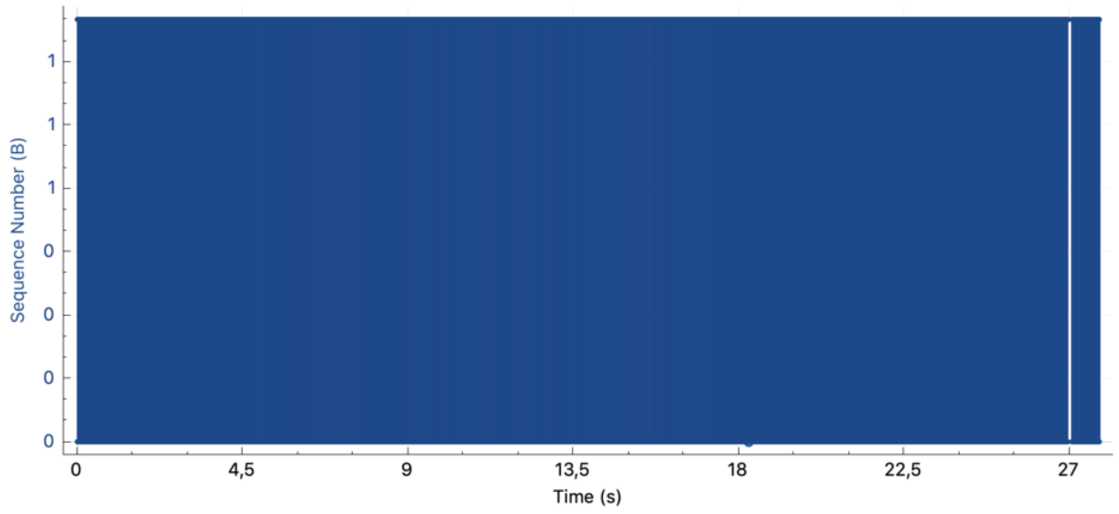
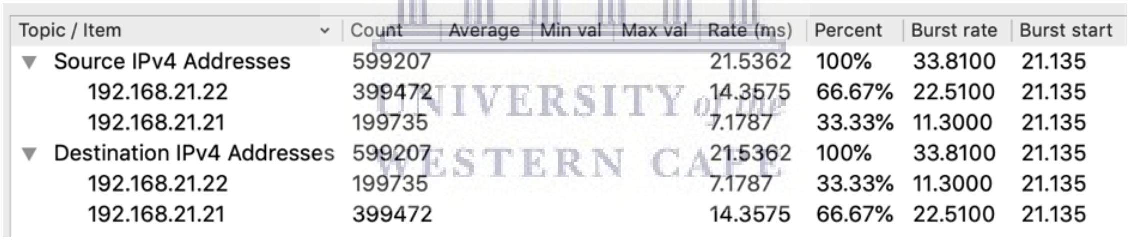


Figure 6.30: TCP horizontal scan - Sequence numbers

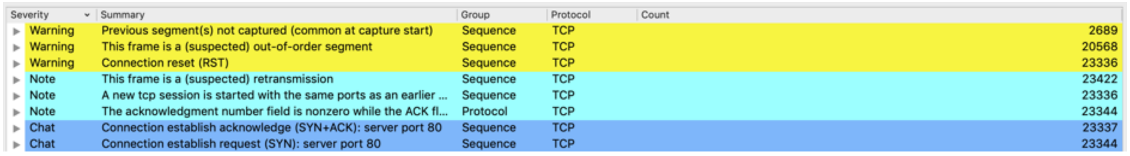
IPv4 source and destination statistics



Topic / Item	Count	Average	Min val	Max val	Rate (ms)	Percent	Burst rate	Burst start
▼ Source IPv4 Addresses	599207				21.5362	100%	33.8100	21.135
192.168.21.22	399472				14.3575	66.67%	22.5100	21.135
192.168.21.21	199735				7.1787	33.33%	11.3000	21.135
▼ Destination IPv4 Addresses	599207				21.5362	100%	33.8100	21.135
192.168.21.22	199735				7.1787	33.33%	11.3000	21.135
192.168.21.21	399472				14.3575	66.67%	22.5100	21.135

Figure 6.31: TCP horizontal scan - IPv4 statistics

Expert Information



Severity	Summary	Group	Protocol	Count
▶ Warning	Previous segment(s) not captured (common at capture start)	Sequence	TCP	2689
▶ Warning	This frame is a (suspected) out-of-order segment	Sequence	TCP	20568
▶ Warning	Connection reset (RST)	Sequence	TCP	23336
▶ Note	This frame is a (suspected) retransmission	Sequence	TCP	23422
▶ Note	A new tcp session is started with the same ports as an earlier ...	Sequence	TCP	23336
▶ Note	The acknowledgment number field is nonzero while the ACK fl...	Protocol	TCP	23344
▶ Chat	Connection establish acknowledge (SYN+ACK): server port 80	Sequence	TCP	23337
▶ Chat	Connection establish request (SYN): server port 80	Sequence	TCP	23344

Figure 6.32: TCP horizontal scan - Expert information

Test capture submitted to a Software as a Service (SaaS) provider for threat analysis scoring

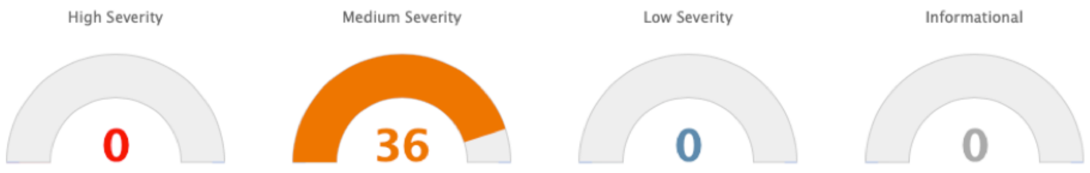


Figure 6.33: TCP horizontal scan - SAAS threat analysis

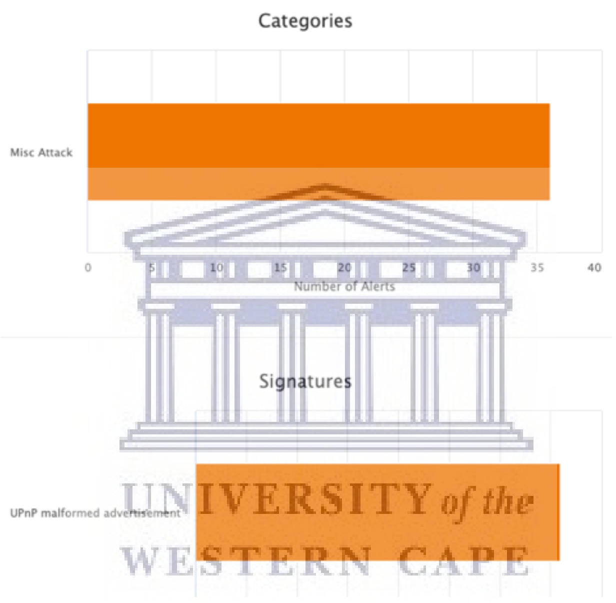
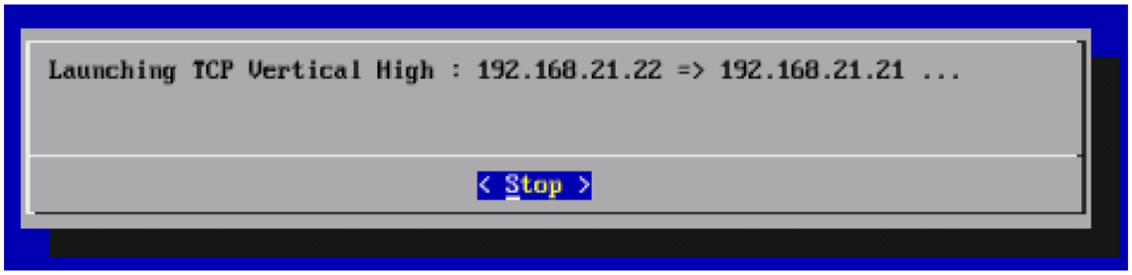


Figure 6.34: TCP horizontal scan - SAAS threat analysis 2

6.0.5 Attack simulation 4 : TCP scan simulation – Type vertical, high volume, layer 4, using ports

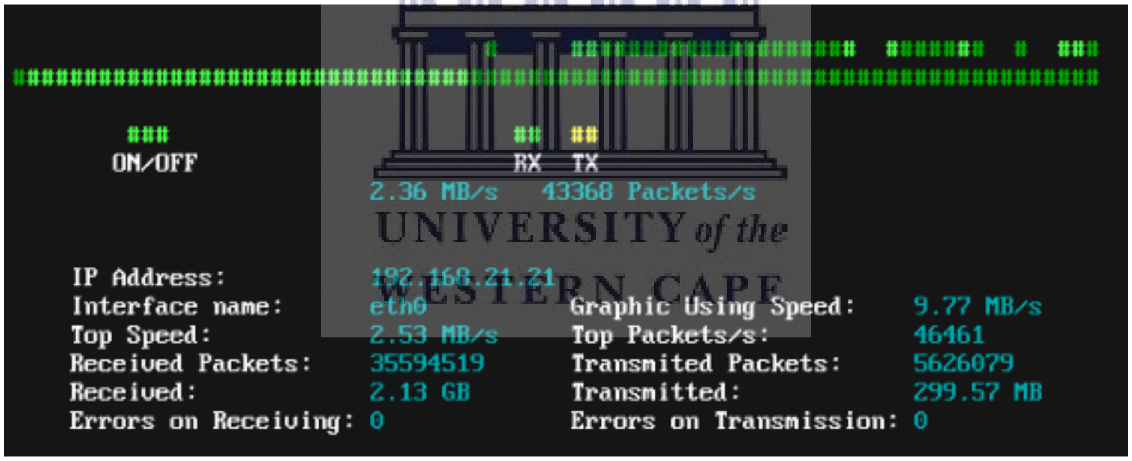
Attack launched



```
Launching TCP Vertical High : 192.168.21.22 => 192.168.21.21 ...
```

Figure 6.35: TCP vertical scan - Attack launched

Victim Statistics

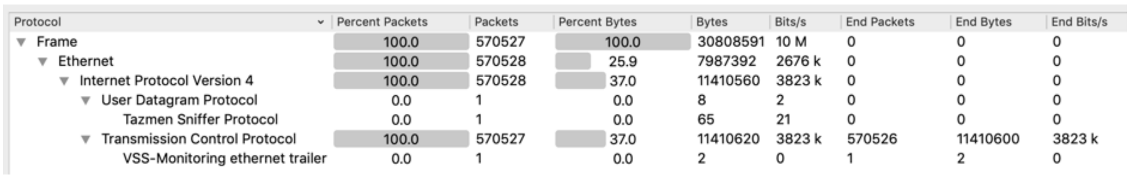


```
        ON/OFF                  RX  TX
                   2.36 MB/s    43368 Packets/s

      IP Address:          192.168.21.21
      Interface name:      eth0          Graphic Using Speed:      9.77 MB/s
      Top Speed:           2.53 MB/s     Top Packets/s:            46461
      Received Packets:    35594519      Transmited Packets:       5626079
      Received:            2.13 GB       Transmitted:              299.57 MB
      Errors on Receiving: 0             Errors on Transmission:   0
```

Figure 6.36: TCP vertical scan - Victim statistics

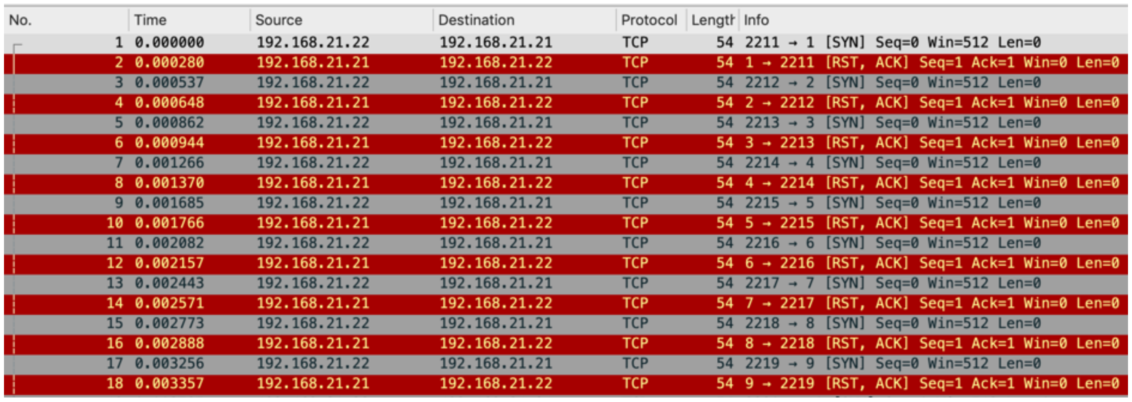
Protocol hierarchy



Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets	End Bytes	End Bits/s
▼ Frame	100.0	570527	100.0	30808591	10 M	0	0	0
▼ Ethernet	100.0	570528	25.9	7987392	2676 k	0	0	0
▼ Internet Protocol Version 4	100.0	570528	37.0	11410560	3823 k	0	0	0
▼ User Datagram Protocol	0.0	1	0.0	8	2	0	0	0
Tazmen Sniffer Protocol	0.0	1	0.0	65	21	0	0	0
▼ Transmission Control Protocol	100.0	570527	37.0	11410620	3823 k	570526	11410600	3823 k
VSS-Monitoring ethernet trailer	0.0	1	0.0	2	0	1	2	0

Figure 6.37: TCP vertical scan - Protocol hierarchy

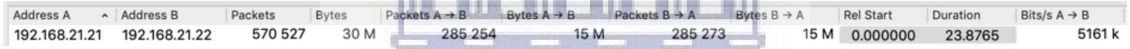
Statistics



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.21.22	192.168.21.21	TCP	54	2211 → 1 [SYN] Seq=0 Win=512 Len=0
2	0.000280	192.168.21.21	192.168.21.22	TCP	54	1 → 2211 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
3	0.000537	192.168.21.22	192.168.21.21	TCP	54	2212 → 2 [SYN] Seq=0 Win=512 Len=0
4	0.000648	192.168.21.21	192.168.21.22	TCP	54	2 → 2212 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
5	0.000862	192.168.21.22	192.168.21.21	TCP	54	2213 → 3 [SYN] Seq=0 Win=512 Len=0
6	0.000944	192.168.21.21	192.168.21.22	TCP	54	3 → 2213 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
7	0.001266	192.168.21.22	192.168.21.21	TCP	54	2214 → 4 [SYN] Seq=0 Win=512 Len=0
8	0.001370	192.168.21.21	192.168.21.22	TCP	54	4 → 2214 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
9	0.001685	192.168.21.22	192.168.21.21	TCP	54	2215 → 5 [SYN] Seq=0 Win=512 Len=0
10	0.001766	192.168.21.21	192.168.21.22	TCP	54	5 → 2215 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
11	0.002082	192.168.21.22	192.168.21.21	TCP	54	2216 → 6 [SYN] Seq=0 Win=512 Len=0
12	0.002157	192.168.21.21	192.168.21.22	TCP	54	6 → 2216 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
13	0.002443	192.168.21.22	192.168.21.21	TCP	54	2217 → 7 [SYN] Seq=0 Win=512 Len=0
14	0.002571	192.168.21.21	192.168.21.22	TCP	54	7 → 2217 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
15	0.002773	192.168.21.22	192.168.21.21	TCP	54	2218 → 8 [SYN] Seq=0 Win=512 Len=0
16	0.002888	192.168.21.21	192.168.21.22	TCP	54	8 → 2218 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
17	0.003256	192.168.21.22	192.168.21.21	TCP	54	2219 → 9 [SYN] Seq=0 Win=512 Len=0
18	0.003357	192.168.21.21	192.168.21.22	TCP	54	9 → 2219 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0

Figure 6.38: TCP vertical scan - Traffic flow capture

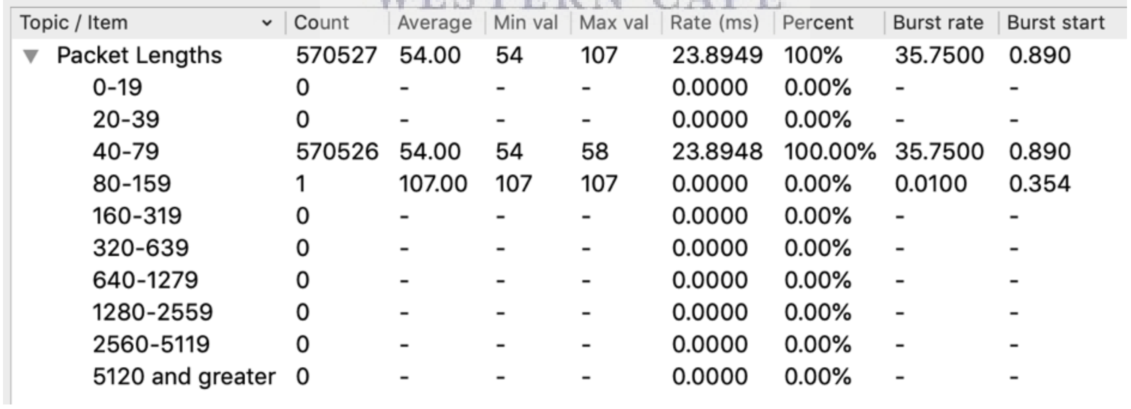
Conversations



Address A	Address B	Packets	Bytes	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start	Duration	Bits/s A → B
192.168.21.21	192.168.21.22	570 527	30 M	285 254	15 M	285 273	15 M	0.000000	23.8765	5161 k

Figure 6.39: TCP vertical scan - Conversations

Packet Lengths



Topic / Item	Count	Average	Min val	Max val	Rate (ms)	Percent	Burst rate	Burst start
▼ Packet Lengths	570527	54.00	54	107	23.8949	100%	35.7500	0.890
0-19	0	-	-	-	0.0000	0.00%	-	-
20-39	0	-	-	-	0.0000	0.00%	-	-
40-79	570526	54.00	54	58	23.8948	100.00%	35.7500	0.890
80-159	1	107.00	107	107	0.0000	0.00%	0.0100	0.354
160-319	0	-	-	-	0.0000	0.00%	-	-
320-639	0	-	-	-	0.0000	0.00%	-	-
640-1279	0	-	-	-	0.0000	0.00%	-	-
1280-2559	0	-	-	-	0.0000	0.00%	-	-
2560-5119	0	-	-	-	0.0000	0.00%	-	-
5120 and greater	0	-	-	-	0.0000	0.00%	-	-

Figure 6.40: TCP vertical scan - Packet lengths

Packets per second graph

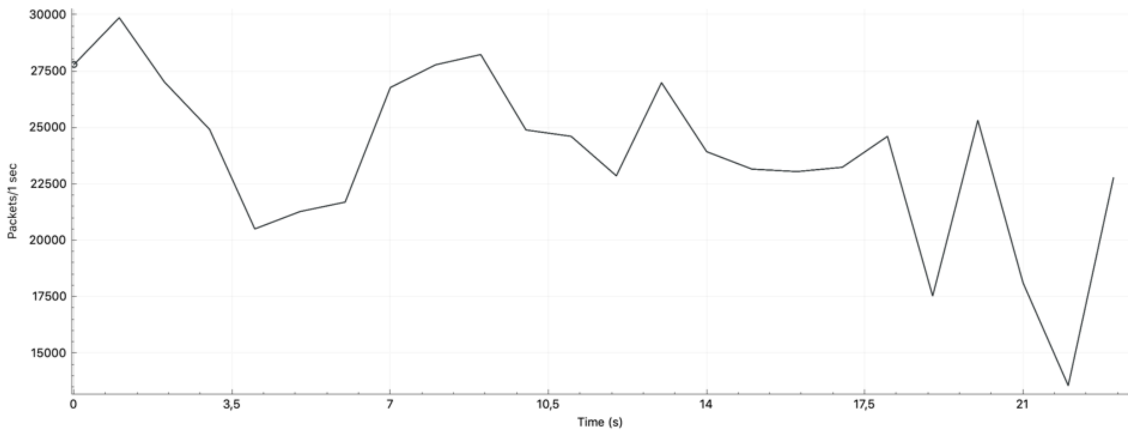


Figure 6.41: TCP vertical scan - Packets per second

Sequence numbers graph

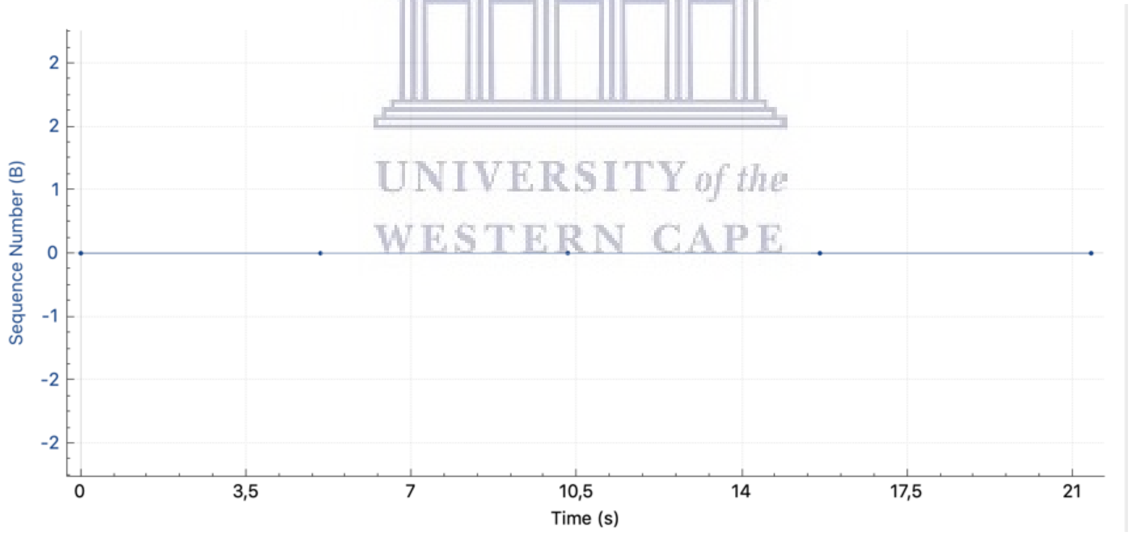
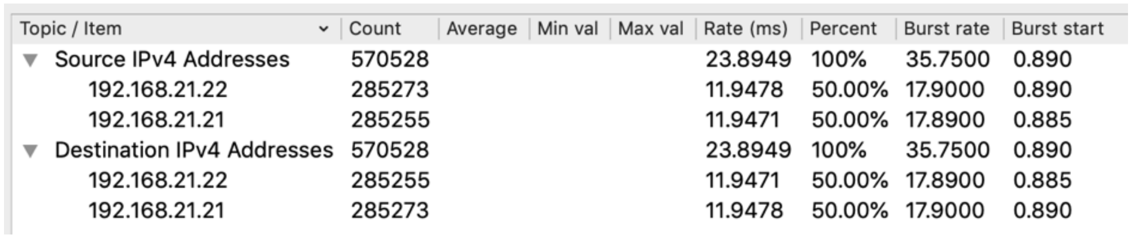


Figure 6.42: TCP vertical scan - Sequence numbers

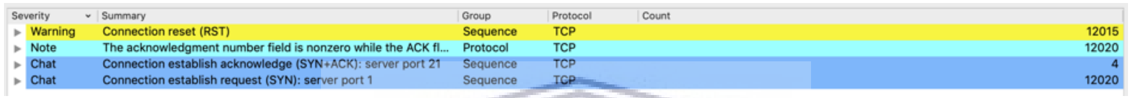
IPv4 source and destination statistics



Topic / Item	Count	Average	Min val	Max val	Rate (ms)	Percent	Burst rate	Burst start
▼ Source IPv4 Addresses	570528				23.8949	100%	35.7500	0.890
192.168.21.22	285273				11.9478	50.00%	17.9000	0.890
192.168.21.21	285255				11.9471	50.00%	17.8900	0.885
▼ Destination IPv4 Addresses	570528				23.8949	100%	35.7500	0.890
192.168.21.22	285255				11.9471	50.00%	17.8900	0.885
192.168.21.21	285273				11.9478	50.00%	17.9000	0.890

Figure 6.43: TCP vertical scan - IPv4 statistics

Expert Information



Severity	Summary	Group	Protocol	Count
▶ Warning	Connection reset (RST)	Sequence	TCP	12015
▶ Note	The acknowledgment number field is nonzero while the ACK fl...	Protocol	TCP	12020
▶ Chat	Connection establish acknowledge (SYN+ACK): server port 21	Sequence	TCP	4
▶ Chat	Connection establish request (SYN): server port 1	Sequence	TCP	12020

Figure 6.44: TCP vertical scan - Expert information

Test capture submitted to SaaS provider for an online threat assessment of the test data

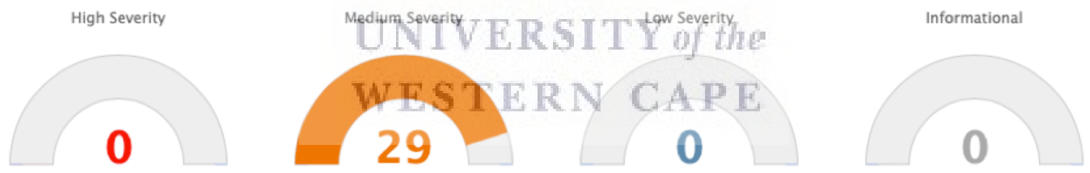


Figure 6.45: TCP vertical scan - SaaS threat analysis part 1

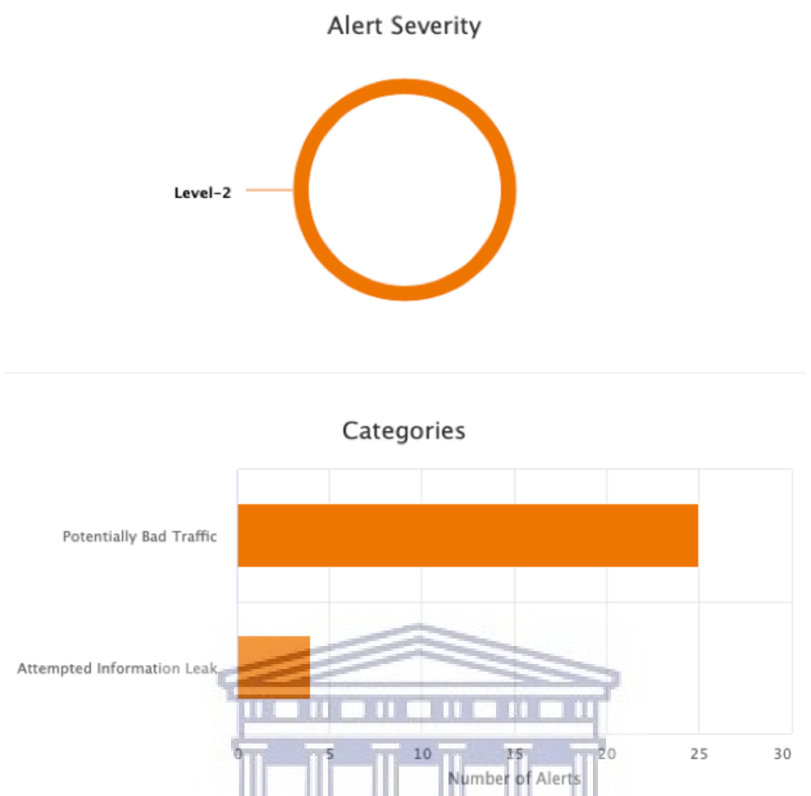


Figure 6.46: TCP vertical scan - SaaS threat analysis part 2

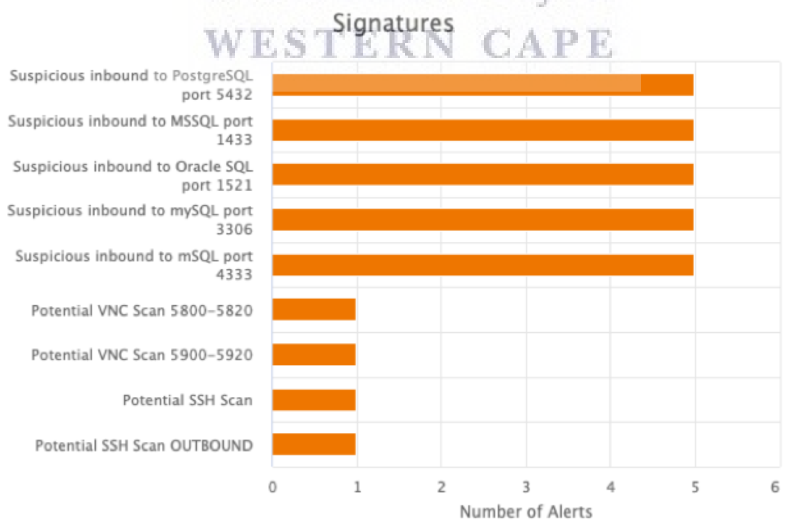
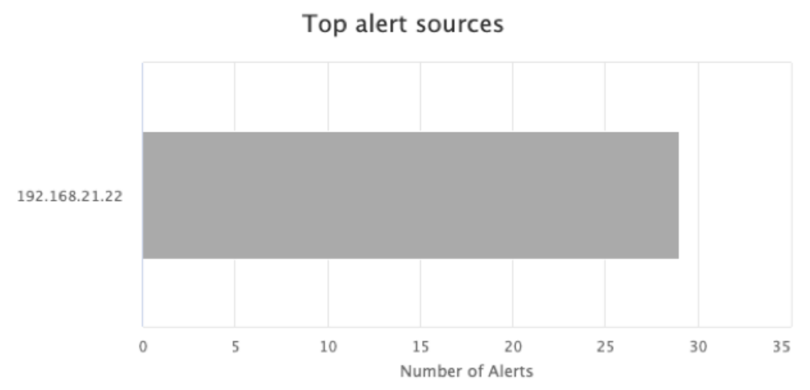


Figure 6.47: TCP vertical scan - SaaS threat analysis part 3



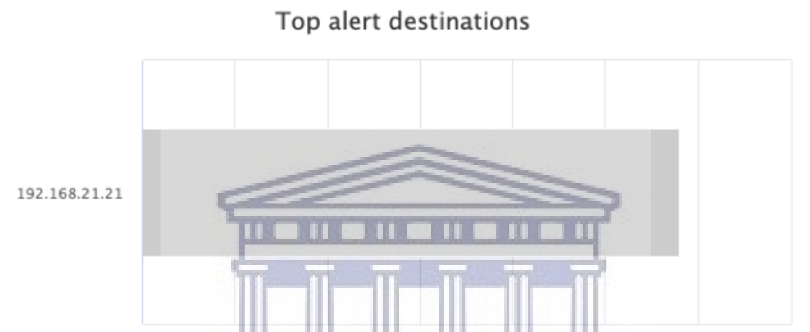
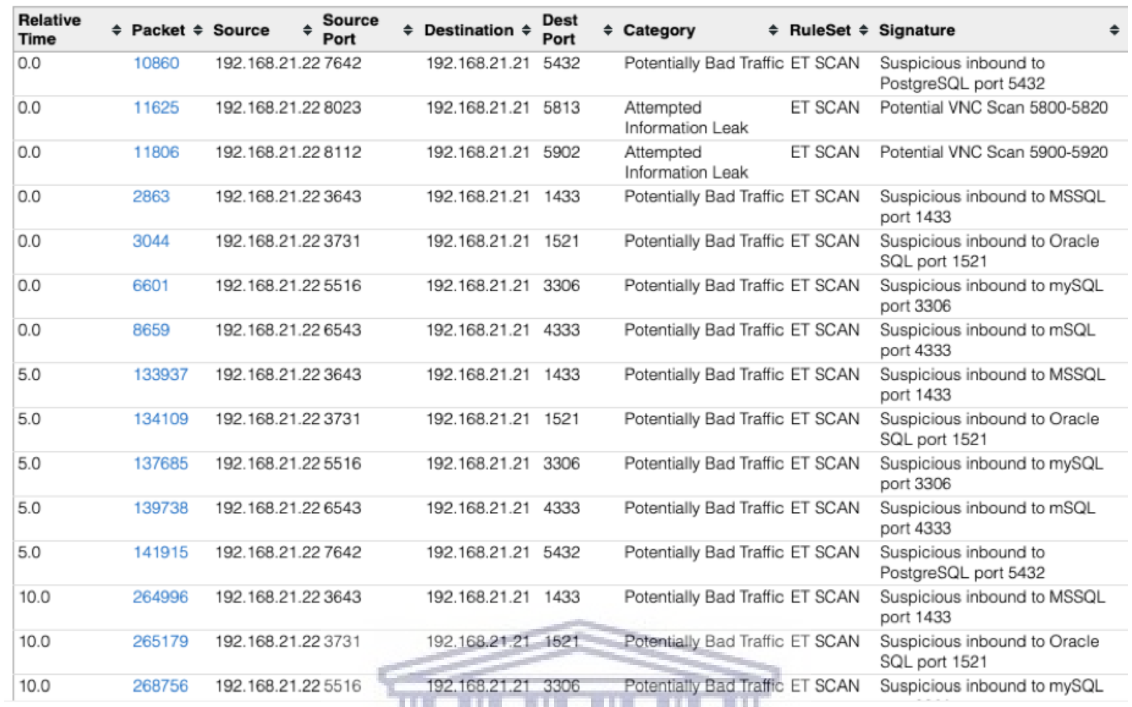


Figure 6.48: TCP vertical scan - SaaS threat analysis part 4

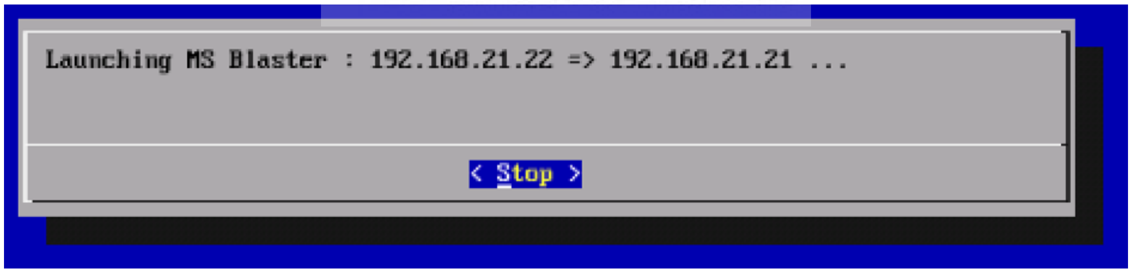


Relative Time	Packet	Source	Source Port	Destination	Dest Port	Category	RuleSet	Signature
0.0	10860	192.168.21.22	7642	192.168.21.21	5432	Potentially Bad Traffic	ET SCAN	Suspicious inbound to PostgreSQL port 5432
0.0	11625	192.168.21.22	8023	192.168.21.21	5813	Attempted Information Leak	ET SCAN	Potential VNC Scan 5800-5820
0.0	11806	192.168.21.22	8112	192.168.21.21	5902	Attempted Information Leak	ET SCAN	Potential VNC Scan 5900-5920
0.0	2863	192.168.21.22	3643	192.168.21.21	1433	Potentially Bad Traffic	ET SCAN	Suspicious inbound to MSSQL port 1433
0.0	3044	192.168.21.22	3731	192.168.21.21	1521	Potentially Bad Traffic	ET SCAN	Suspicious inbound to Oracle SQL port 1521
0.0	6601	192.168.21.22	5516	192.168.21.21	3306	Potentially Bad Traffic	ET SCAN	Suspicious inbound to mySQL port 3306
0.0	8659	192.168.21.22	6543	192.168.21.21	4333	Potentially Bad Traffic	ET SCAN	Suspicious inbound to mSQL port 4333
5.0	133937	192.168.21.22	3643	192.168.21.21	1433	Potentially Bad Traffic	ET SCAN	Suspicious inbound to MSSQL port 1433
5.0	134109	192.168.21.22	3731	192.168.21.21	1521	Potentially Bad Traffic	ET SCAN	Suspicious inbound to Oracle SQL port 1521
5.0	137685	192.168.21.22	5516	192.168.21.21	3306	Potentially Bad Traffic	ET SCAN	Suspicious inbound to mySQL port 3306
5.0	139738	192.168.21.22	6543	192.168.21.21	4333	Potentially Bad Traffic	ET SCAN	Suspicious inbound to mSQL port 4333
5.0	141915	192.168.21.22	7642	192.168.21.21	5432	Potentially Bad Traffic	ET SCAN	Suspicious inbound to PostgreSQL port 5432
10.0	264996	192.168.21.22	3643	192.168.21.21	1433	Potentially Bad Traffic	ET SCAN	Suspicious inbound to MSSQL port 1433
10.0	265179	192.168.21.22	3731	192.168.21.21	1521	Potentially Bad Traffic	ET SCAN	Suspicious inbound to Oracle SQL port 1521
10.0	268756	192.168.21.22	5516	192.168.21.21	3306	Potentially Bad Traffic	ET SCAN	Suspicious inbound to mySQL

Figure 6.49: TCP vertical scan - SaaS threat analysis part 5

6.0.6 Attack Simulations : Worm propagation simulation | MS Blaster

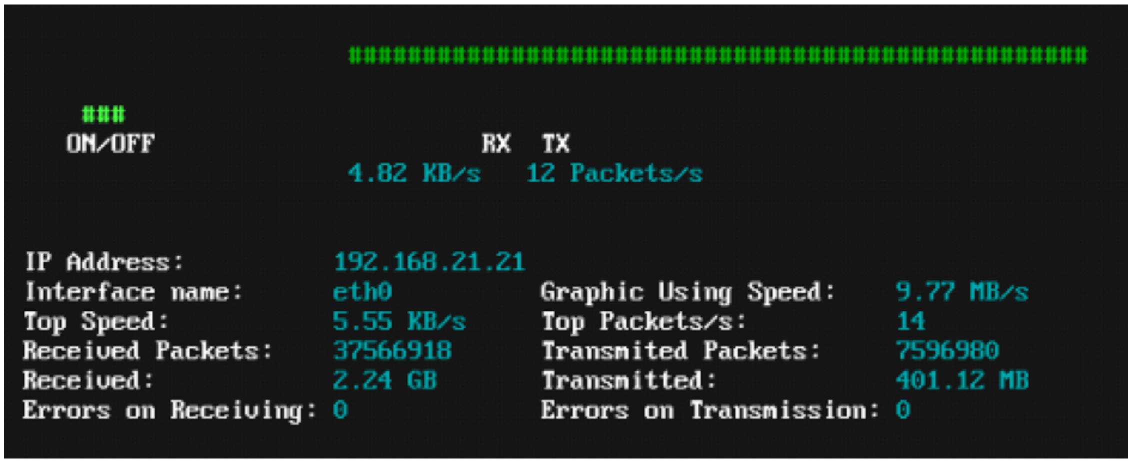
Attack launched



```
Launching MS Blaster : 192.168.21.22 => 192.168.21.21 ...
```

Figure 6.50: Worm propagation - Attack launched

Victim statistics



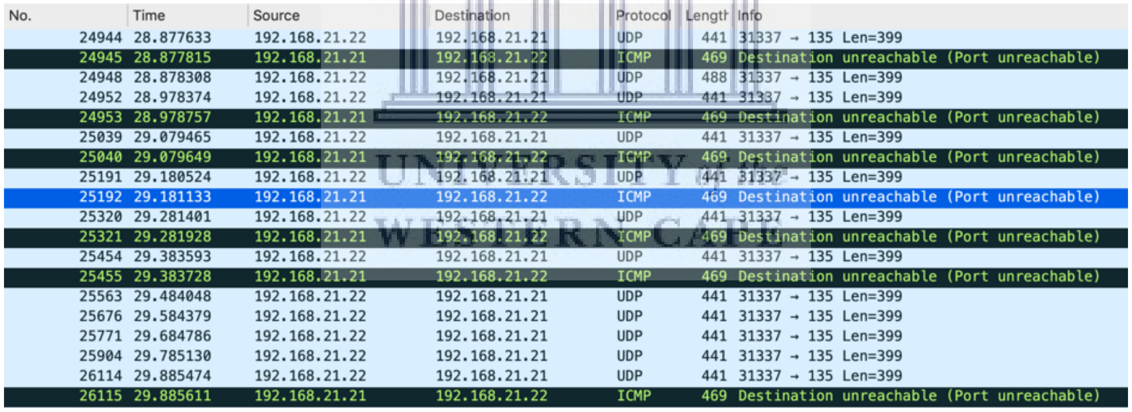
```
                ##################################################

   ###
  ON/OFF                        RX   TX
                4.82 KB/s    12 Packets/s

IP Address:             192.168.21.21
Interface name:         eth0            Graphic Using Speed:     9.77 MB/s
Top Speed:              5.55 KB/s       Top Packets/s:           14
Received Packets:       37566918        Transmited Packets:      7596980
Received:               2.24 GB         Transmitted:             401.12 MB
Errors on Receiving:    0               Errors on Transmission:  0
```

Figure 6.51: Worm propagation - Victim statistics

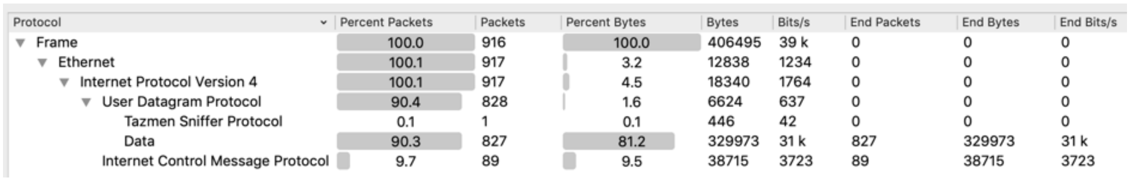
Traffic flow capture



No.	Time	Source	Destination	Protocol	Length	Info
24944	28.877633	192.168.21.22	192.168.21.21	UDP	441	31337 → 135 Len=399
24945	28.877815	192.168.21.21	192.168.21.22	ICMP	469	Destination unreachable (Port unreachable)
24948	28.878308	192.168.21.22	192.168.21.21	UDP	488	31337 → 135 Len=399
24952	28.978374	192.168.21.22	192.168.21.21	UDP	441	31337 → 135 Len=399
24953	28.978757	192.168.21.21	192.168.21.22	ICMP	469	Destination unreachable (Port unreachable)
25039	29.079465	192.168.21.22	192.168.21.21	UDP	441	31337 → 135 Len=399
25040	29.079649	192.168.21.21	192.168.21.22	ICMP	469	Destination unreachable (Port unreachable)
25191	29.180524	192.168.21.22	192.168.21.21	UDP	441	31337 → 135 Len=399
25192	29.181133	192.168.21.21	192.168.21.22	ICMP	469	Destination unreachable (Port unreachable)
25320	29.281401	192.168.21.22	192.168.21.21	UDP	441	31337 → 135 Len=399
25321	29.281928	192.168.21.21	192.168.21.22	ICMP	469	Destination unreachable (Port unreachable)
25454	29.383593	192.168.21.22	192.168.21.21	UDP	441	31337 → 135 Len=399
25455	29.383728	192.168.21.21	192.168.21.22	ICMP	469	Destination unreachable (Port unreachable)
25563	29.484048	192.168.21.22	192.168.21.21	UDP	441	31337 → 135 Len=399
25676	29.584379	192.168.21.22	192.168.21.21	UDP	441	31337 → 135 Len=399
25771	29.684786	192.168.21.22	192.168.21.21	UDP	441	31337 → 135 Len=399
25904	29.785130	192.168.21.22	192.168.21.21	UDP	441	31337 → 135 Len=399
26114	29.885474	192.168.21.22	192.168.21.21	UDP	441	31337 → 135 Len=399
26115	29.885611	192.168.21.21	192.168.21.22	ICMP	469	Destination unreachable (Port unreachable)

Figure 6.52: Worm propagation - Traffic flow capture

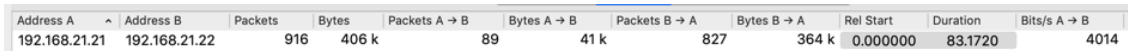
Protocol hierarchy



Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets	End Bytes	End Bits/s
▼ Frame	100.0	916	100.0	406495	39 k	0	0	0
▼ Ethernet	100.1	917	3.2	12838	1234	0	0	0
▼ Internet Protocol Version 4	100.1	917	4.5	18340	1764	0	0	0
▼ User Datagram Protocol	90.4	828	1.6	6624	637	0	0	0
Tazmen Sniffer Protocol	0.1	1	0.1	446	42	0	0	0
Data	90.3	827	81.2	329973	31 k	827	329973	31 k
Internet Control Message Protocol	9.7	89	9.5	38715	3723	89	38715	3723

Figure 6.53: Worm propagation - Protocol hierarchy

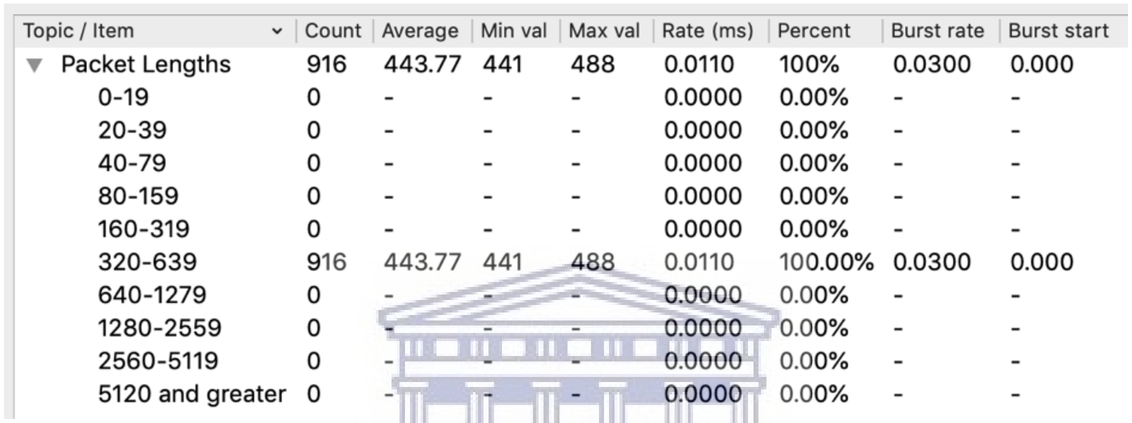
Conversations



Address A	Address B	Packets	Bytes	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start	Duration	Bits/s A → B
192.168.21.21	192.168.21.22	916	406 k	89	41 k	827	364 k	0.000000	83.1720	4014

Figure 6.54: Worm propagation - Conversations

Packet Lengths



Topic / Item	Count	Average	Min val	Max val	Rate (ms)	Percent	Burst rate	Burst start
▼ Packet Lengths	916	443.77	441	488	0.0110	100%	0.0300	0.000
0-19	0	-	-	-	0.0000	0.00%	-	-
20-39	0	-	-	-	0.0000	0.00%	-	-
40-79	0	-	-	-	0.0000	0.00%	-	-
80-159	0	-	-	-	0.0000	0.00%	-	-
160-319	0	-	-	-	0.0000	0.00%	-	-
320-639	916	443.77	441	488	0.0110	100.00%	0.0300	0.000
640-1279	0	-	-	-	0.0000	0.00%	-	-
1280-2559	0	-	-	-	0.0000	0.00%	-	-
2560-5119	0	-	-	-	0.0000	0.00%	-	-
5120 and greater	0	-	-	-	0.0000	0.00%	-	-

Figure 6.55: Worm propagation - Packets lengths

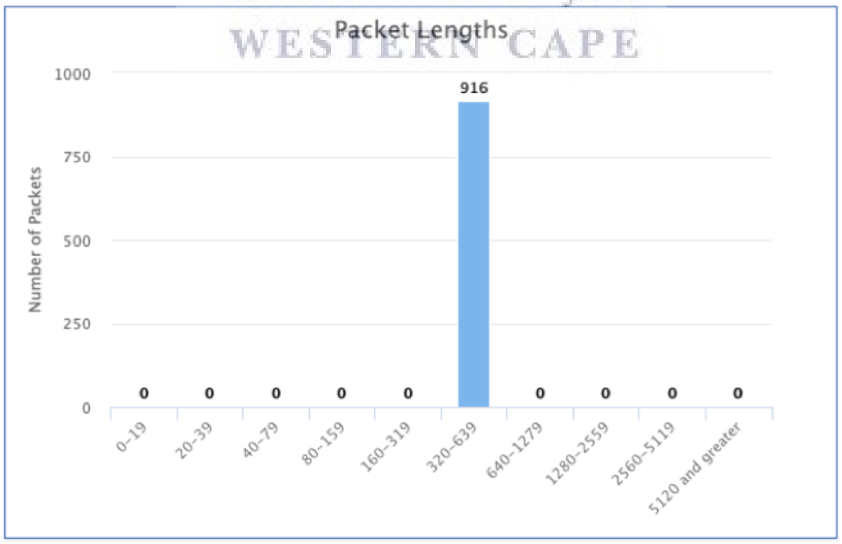


Figure 6.56: Worm propagation - Packet length graph

Packets Per Second Graph

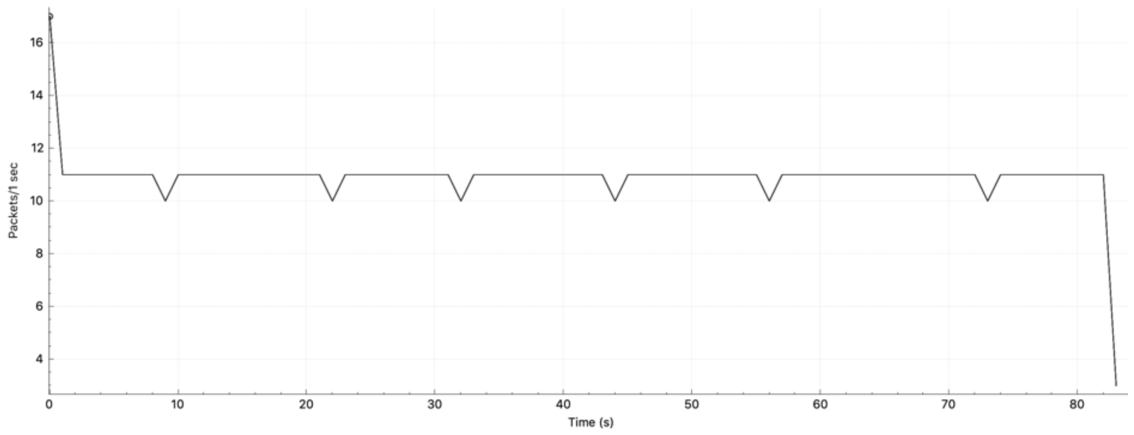
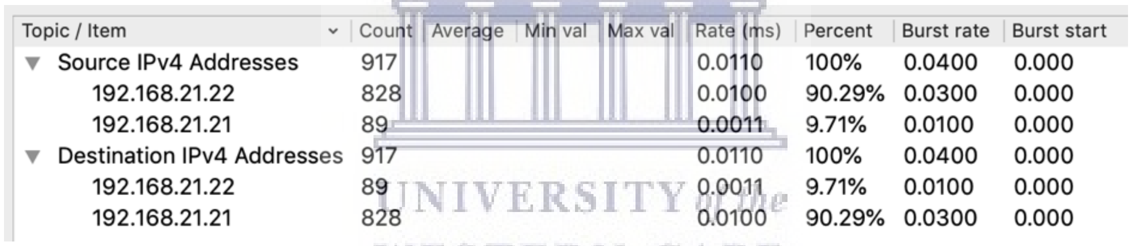


Figure 6.57: Worm propagation - Packets per second

IPv4 source and destination statistics

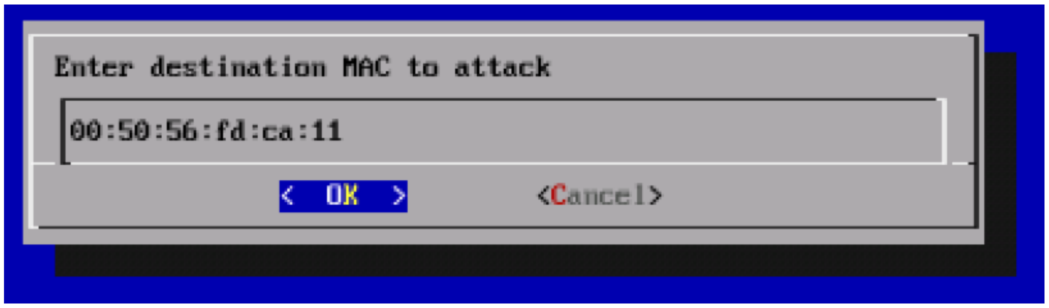


Topic / Item	Count	Average	Min val	Max val	Rate (ms)	Percent	Burst rate	Burst start
▼ Source IPv4 Addresses	917				0.0110	100%	0.0400	0.000
192.168.21.22	828				0.0100	90.29%	0.0300	0.000
192.168.21.21	89				0.0011	9.71%	0.0100	0.000
▼ Destination IPv4 Addresses	917				0.0110	100%	0.0400	0.000
192.168.21.22	89				0.0011	9.71%	0.0100	0.000
192.168.21.21	828				0.0100	90.29%	0.0300	0.000

Figure 6.58: Worm propagation - IPv4 statistics

6.0.7 Attack simulation : Intrusion detection – HTTP URI fragment and max attack

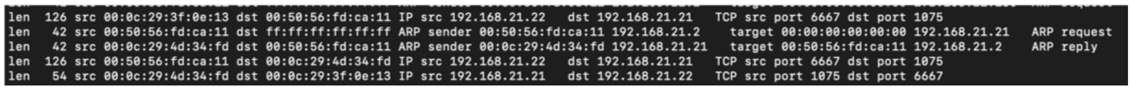
MAC address attack simulation



```
Enter destination MAC to attack
00:50:56:fd:ca:11
```

Figure 6.59: Intrusion detection - MAC address

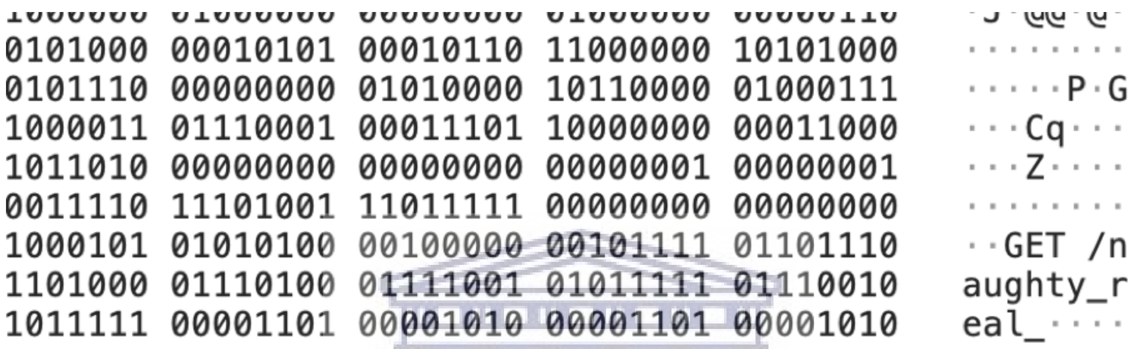
MAC address exploitation



```
len  126 src 00:0c:29:3f:0e:13 dst 00:50:56:fd:ca:11 IP src 192.168.21.22   dst 192.168.21.21   TCP src port 6667 dst port 1075
len   42 src 00:50:56:fd:ca:11 dst ff:ff:ff:ff:ff:ff ARP sender 00:50:56:fd:ca:11 192.168.21.2    target 00:00:00:00:00:00 192.168.21.21   ARP request
len   42 src 00:0c:29:4d:34:fd dst 00:50:56:fd:ca:11 ARP sender 00:0c:29:4d:34:fd 192.168.21.21   target 00:50:56:fd:ca:11 192.168.21.2    ARP reply
len  126 src 00:50:56:fd:ca:11 dst 00:0c:29:4d:34:fd IP src 192.168.21.22   dst 192.168.21.21   TCP src port 6667 dst port 1075
len   54 src 00:0c:29:4d:34:fd dst 00:0c:29:3f:0e:13 IP src 192.168.21.21   dst 192.168.21.22   TCP src port 1075 dst port 6667
```

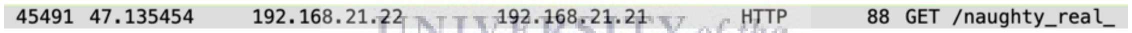
Figure 6.60: Intrusion detection - MAC exploit

Traffic capture



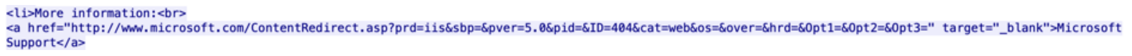
```
0101000 00010101 00010110 11000000 10101000   ········
0101110 00000000 01010000 10110000 01000111   ·····P·G
1000011 01110001 00011101 10000000 00011000   ···Cq···
1011010 00000000 00000000 00000001 00000001   ···Z····
0011110 11101001 11011111 00000000 00000000   ········
1000101 01010100 00100000 00101111 01101110   ··GET /n
1101000 01110100 01111001 01011111 01110010   aughty_r
1011111 00001101 00001010 00001101 00001010   eal_····
```

Figure 6.61: Intrusion detection - Traffic capture



```
45491  47.135454    192.168.21.22    192.168.21.21    HTTP    88  GET /naughty_real_
```

Figure 6.62: Intrusion detection - Payload detail part 1



```
<li>More information:<br>
<a href="http://www.microsoft.com/ContentRedirect.asp?prd=iis&sbp=&pver=5.0&pid=&ID=404&cat=web&os=&over=&hrd=&Opt1=&Opt2=&Opt3=" target="_blank">Microsoft
Support</a>
```

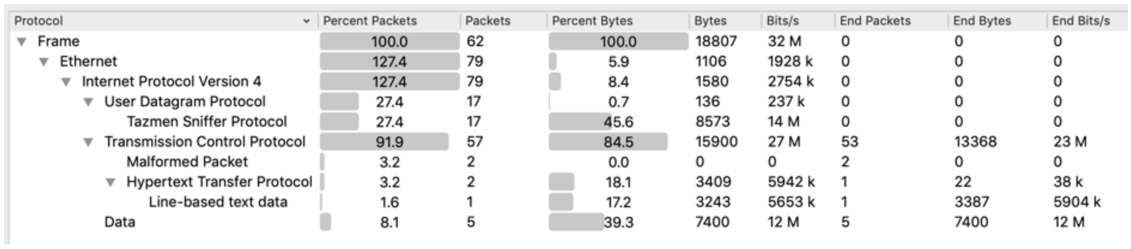
Figure 6.63: Intrusion detection - Payload detail part 2

Hacking IIS 5 and Web applications reference

```
GET /naughty_real_ - 404 GET /scripts/sensepost.exe /c+echo*␣
 ↪*Olifante%20onder%20my%20bed* *sensepost.exe* POST /scripts/upload.asp - 200␣
 ↪POST /scripts/cmdasp.asp - 200 POST /scripts/cmdasp.asp␣
 ↪|-|ASP_0113|Script_timed_out 500
```

```
https://epdf.pub/
↪hacking-exposed-network-security-secrets-amp-solutions-third-edition-hacking-exp9b39ce614100f
↪html
```

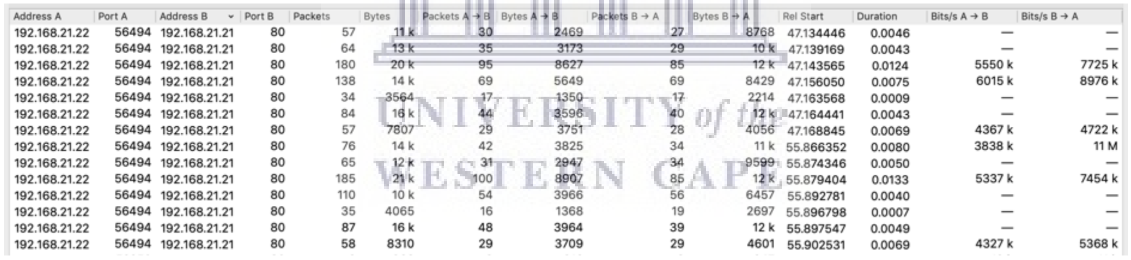
Protocol hierarchy



Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets	End Bytes	End Bits/s
▼ Frame	100.0	62	100.0	18807	32 M	0	0	0
▼ Ethernet	127.4	79	5.9	1106	1928 k	0	0	0
▼ Internet Protocol Version 4	127.4	79	8.4	1580	2754 k	0	0	0
▼ User Datagram Protocol	27.4	17	0.7	136	237 k	0	0	0
Tazmen Sniffer Protocol	27.4	17	45.6	8573	14 M	0	0	0
▼ Transmission Control Protocol	91.9	57	84.5	15900	27 M	53	13368	23 M
Malformed Packet	3.2	2	0.0	0	0	2	0	0
▼ Hypertext Transfer Protocol	3.2	2	18.1	3409	5942 k	1	22	38 k
Line-based text data	1.6	1	17.2	3243	5653 k	1	3387	5904 k
Data	8.1	5	39.3	7400	12 M	5	7400	12 M

Figure 6.64: Intrusion detection - Protocol hierarchy

Conversations



Address A	Port A	Address B	Port B	Packets	Bytes	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start	Duration	Bits/s A → B	Bits/s B → A
192.168.21.22	56494	192.168.21.21	80	57	11 k	30	2469	27	8768	47.134446	0.0046	—	—
192.168.21.22	56494	192.168.21.21	80	64	13 k	35	3173	29	10 k	47.139169	0.0043	—	—
192.168.21.22	56494	192.168.21.21	80	180	20 k	95	8627	85	12 k	47.143565	0.0124	5550 k	7725 k
192.168.21.22	56494	192.168.21.21	80	138	14 k	69	5649	69	8429	47.156050	0.0075	6015 k	8976 k
192.168.21.22	56494	192.168.21.21	80	34	3564	17	1350	17	2214	47.163568	0.0009	—	—
192.168.21.22	56494	192.168.21.21	80	84	16 k	44	3596	40	12 k	47.164441	0.0043	—	—
192.168.21.22	56494	192.168.21.21	80	57	7807	29	3751	28	4056	47.168845	0.0069	4367 k	4722 k
192.168.21.22	56494	192.168.21.21	80	76	14 k	42	3825	34	11 k	55.866352	0.0080	3838 k	11 M
192.168.21.22	56494	192.168.21.21	80	65	12 k	31	2947	34	9599	55.874346	0.0050	—	—
192.168.21.22	56494	192.168.21.21	80	185	21 k	100	8907	85	12 k	55.879404	0.0133	5337 k	7454 k
192.168.21.22	56494	192.168.21.21	80	110	10 k	54	3966	56	6457	55.892781	0.0040	—	—
192.168.21.22	56494	192.168.21.21	80	35	4065	16	1368	19	2697	55.896798	0.0007	—	—
192.168.21.22	56494	192.168.21.21	80	87	16 k	48	3964	39	12 k	55.897547	0.0049	—	—
192.168.21.22	56494	192.168.21.21	80	58	8310	29	3709	29	4601	55.902531	0.0069	4327 k	5368 k

Figure 6.65: Intrusion detection - Conversations

Packets Per Second Graph

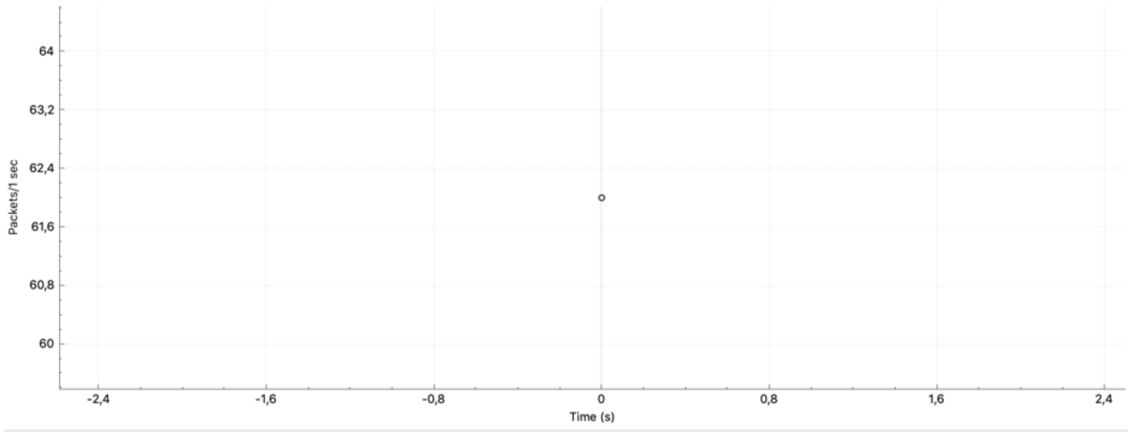


Figure 6.66: Intrusion detection - Packets per second

Sequence Numbers Graph

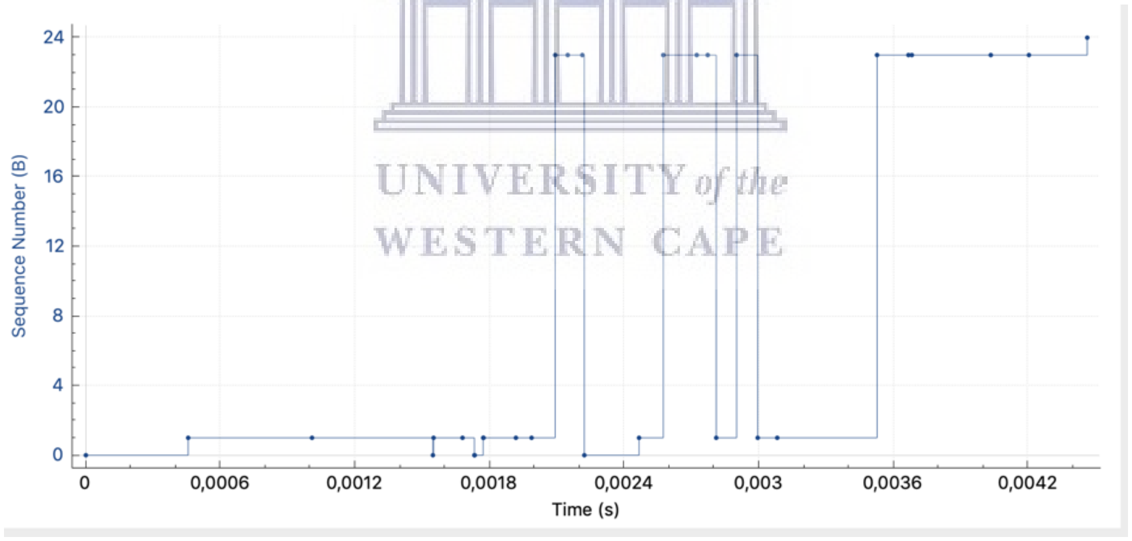
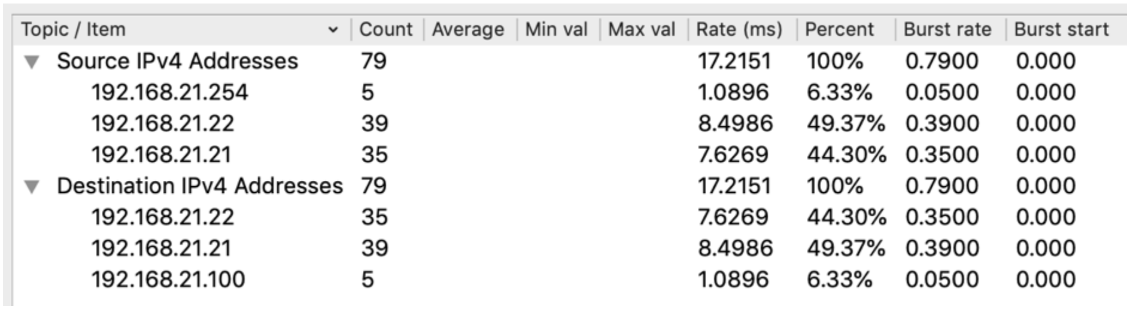


Figure 6.67: Intrusion detection - Sequence Numbers

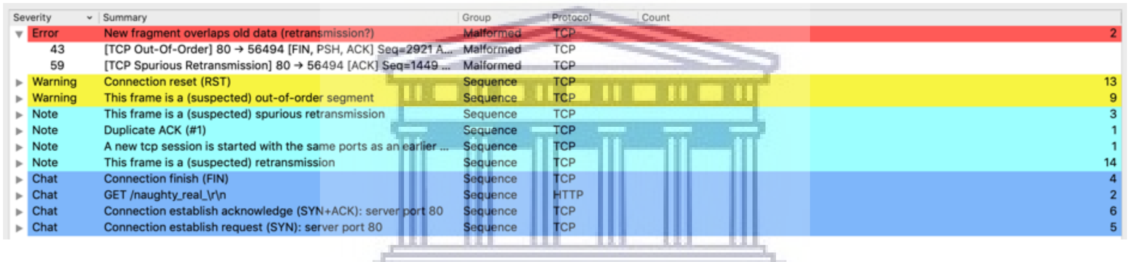
IPv4 source and destination statistics



Topic / Item	Count	Average	Min val	Max val	Rate (ms)	Percent	Burst rate	Burst start
▼ Source IPv4 Addresses	79				17.2151	100%	0.7900	0.000
192.168.21.254	5				1.0896	6.33%	0.0500	0.000
192.168.21.22	39				8.4986	49.37%	0.3900	0.000
192.168.21.21	35				7.6269	44.30%	0.3500	0.000
▼ Destination IPv4 Addresses	79				17.2151	100%	0.7900	0.000
192.168.21.22	35				7.6269	44.30%	0.3500	0.000
192.168.21.21	39				8.4986	49.37%	0.3900	0.000
192.168.21.100	5				1.0896	6.33%	0.0500	0.000

Figure 6.68: Intrusion detection - IPv4 statistics

Expert Information



Severity	Summary	Group	Protocol	Count
▼ Error	New fragment overlaps old data (retransmission?)	Malformed	TCP	2
43	[TCP Out-Of-Order] 80 → 56494 [FIN, PSH, ACK] Seq=2921 A...	Malformed	TCP	
59	[TCP Spurious Retransmission] 80 → 56494 [ACK] Seq=1449 ...	Malformed	TCP	
▶ Warning	Connection reset (RST)	Sequence	TCP	13
▶ Warning	This frame is a (suspected) out-of-order segment	Sequence	TCP	9
▶ Note	This frame is a (suspected) spurious retransmission	Sequence	TCP	3
▶ Note	Duplicate ACK (#1)	Sequence	TCP	1
▶ Note	A new tcp session is started with the same ports as an earlier ...	Sequence	TCP	1
▶ Note	This frame is a (suspected) retransmission	Sequence	TCP	14
▶ Chat	Connection finish (FIN)	Sequence	TCP	4
▶ Chat	GET /naughty_real_\r\n	Sequence	HTTP	2
▶ Chat	Connection establish acknowledge (SYN+ACK): server port 80	Sequence	TCP	6
▶ Chat	Connection establish request (SYN): server port 80	Sequence	TCP	5

Figure 6.69: Intrusion detection - Expert information

Bibliography

[1] Mohammad NT Elbatta. An improvement for dbscan algorithm for best results in varied densities. *An Improvement for DBSCAN Algorithm for Best Results in Varied Densities*, 2012.

[2] Zhanyi Wang. The applications of deep learning on traffic identification. *BlackHat USA*, 24(11):1–10, 2015.

[3] Ryan Gavin Goss. *APIC: A method for automated pattern identification and classification*. PhD thesis, University of Cape Town, 2017.

[4] K. Xu, Z. L. Zhang, and S. Bhattacharyya. Profiling internet backbone traffic: behavior models and applications. *In ACM SIGCOMM Computer Communication Review*, 35(4):169–180, 2005.

[5] Guoqiang Zhong, Hui Xu, Pan Yang, Sijiang Wang, and Junyu Dong. Deep hashing learning networks. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 2236–2243. IEEE, 2016.

[6] T. T. Nguyen and G. Armitage. Training on multiple sub-flows to optimise the use of machine learning classifiers in real-world ip networks. *In Local Computer Networks, Proceedings 2006 31st, IEEE Conference*, 31:369–376, 2006.

[7] T. T. Nguyen and G. Armitage. A survey of techniques for internet traffic classification using machine learning. *Communications Surveys and Tutorials, 10(4)*, pages 56–76., 2008.

[8] I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques.* Morgan Kaufmann, 2005.

[9] Sebastian Zander, Thuy Nguyen, and Grenville Armitage. Automated traffic classification and application identification using machine learning. In *The IEEE Conference on Local Computer Networks 30th Anniversary (LCN'05) l*, pages 250–257. IEEE, 2005.

[10] Laurent Bernaille, Renata Teixeira, Ismael Akodkenou, Augustin Soule, and Kave Salamatian. Traffic classification on the fly. *ACM SIGCOMM Computer Communication Review*, 36(2):23–26, 2006.

[11] T. Auld, A. W. Moore, and S. F. Gull. Bayesian neural networks for internet traffic classification. neural networks, transactions on, 18(1). *Nueral Networks*, 18(1):223–239, 2007.

[12] Yudha Purwanto, Budi Rahardjo, et al. Traffic anomaly detection in ddos flooding attack. In *2014 8th International Conference on Telecommunication Systems Services and Applications (TSSA)*, pages 1–6. IEEE, 2014.

[13] D. Wang, L. Zhang, Y. Z. Long, Y. B. Xue, and Y. F. Dong. Application behavior characterization (abc) system for fast and accurate large scale traffic classification, 2010.

[14] Manuel Alberto M Ferreira and Marina Andrade. Fundaments of theory of queues. *International Journal of Academic Research*, 3(1 Part II):427–429, 2011.

[15] Sean P Meyn and Richard L Tweedie. *Markov chains and stochastic stability*. Springer Science & Business Media, 2012.

[16] Olasupo Ajayi, Antoine Bagula, Ifeoma Chukwubueze, and Hloniphani Maluleke. Priority based traffic pre-emption system for medical emergency vehicles in smart cities. In *2020 IEEE Symposium on Computers and Communications (ISCC)*, pages 1–7, 2020.

[17] Olasupo Ajayi, Antoine Bagula, Omowunmi Isafiade, and Ayodele Noutouglo. *Effective Management of Delays at Road Intersections Using Smart Traffic Light System*. Springer, 2020.

[18] Anurag Sahu, Ashish Garg, and Ambesh Dixit. A review on quantum dot sensitized solar cells: Past, present and future towards carrier multiplication with a possibility for higher efficiency. *Solar Energy*, 203:210–239, 2020.

[19] Narmeen Zakaria Bawany, Jawwad A Shamsi, and Khaled Salah. Ddos attack detection and mitigation using sdn: methods, practices, and solutions. *Arabian Journal for Science and Engineering*, 42(2):425–441, 2017.

[20] Noor Zuraidin Mohd Safar, Noryusliza Abdullah, Hazalila Kamaludin, Suhaimi Abd Ishak, and Mohd Rizal Mohd Isa. Characterising and detection of botnet in p2p network for udp

protocol. *Indonesian Journal of Electrical Engineering and Computer Science*, 18(3):1584–1595, 2020.

[21] J Siwek. https://github.com/jsiwek/capsan. *Capsan*, 2019.

[22] Thuy TT Nguyen and Grenville Armitage. Training on multiple sub-flows to optimise the use of machine learning classifiers in real-world ip networks. In *Local Computer Networks, Proceedings 2006 31st IEEE Conference on*, pages 369–376. IEEE, 2006.

[23] Mark W Isken. An optimization model based decision support system for staff scheduling analysis in healthcare facilities. *AMCIS 2000 Proceedings*, page 154, 2000.

[24] Pierre Le Gall et al. Single server queueing networks with varying service times and renewal input. *International Journal of Stochastic Analysis*, 13:1–22, 2000.