DETECTING MALICIOUS DNS TUNNELS VIA NETWORK FLOW ENTROPY

by

Yulduz Khodjaeva

Submitted in partial fulfillment of the requirements
for the degree of Master of Computer Science

at

Dalhousie University
Halifax, Nova Scotia
December 2021

The thesis is dedicated to my parents and to all people who have been with me during these challenging times

Table of Contents

List of Tables

List of Figures

Abstract

The thesis proposes the concept of "entropy of a flow" to augment flow statistical features for DNS tunnelling detection, specifically DNS over HTTPS traffic. To achieve this, the use of flow exporters, namely Argus, DoHlyzer and Tranalyzer2 are explored. Flow features are then augmented with the flow entropy, calculated in three different ways: entropy over all packets of a flow, entropy over the first 96 bytes of a flow, entropy over the first n-packets of a flow. These features are provided as input to five machine learning classifiers, specifically Decision Tree, Random Forest, Logistic Regression, Support Vector Machine and Naive Bayes to detect malicious behaviours in different publicly available datasets. Evaluations show that the Decision Tree algorithm could reach an F-measure of approximately 99.7% when flow statistical features are augmented with the flow entropy of the first four packets. This model is then optimized using TPOT-AutoML, where the Random Forest classifier provided the best pipeline configuration for the same features.

List of Abbreviations Used

ARP	Address Resolution Protocol
AutoAI	Automated Artificial Intelligence
AUC	Area Under Curve
CNN	Convolutional Neural Network
DNS	Domain Name System
DoH	DNS over HTTPS
DoT	DNS over TLS
DT	Decision Tree
DoS	Denial of Service
DL	Deep Learning
ESP	Encapsulating Security Protocol
FTP	File Transfer Protocol
GB	Gradient Boosting
GML	General Machine Learning
GUI	Graphical User Interface
GP	Genetic Programming
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
ICMP	Internet Control Message Protocol
IETF	Internet Engineering Task Force
IoT	Internet of Things
IP	Internet Protocol
IDS	Intrusion Detection System
KNN	k-nearest neighbor
LGBM	Light Gradient Boosting Machine
LR	Logistic Regression
ML	Machine Learning
NB	Naive Bayes

NLP	Natural Language Processing
PCC	Pearson Correlation Coefficient
PC	Personal Computer
RF	Random Forest
SSH	Secure Shell
SOM	Self-Organizing Map
SVM	Support Vector Machine
TLS	Transport Layer Security
T2	Tranalyzer2
URL	Uniform Resource Locator
VANET	Vehicular Ad hoc Network
WEKA	Waikato Environment for Knowledge Analysis
XGBM	Extreme Gradient Boosting Machine

Acknowledgements

I would like to express my gratitude to my supervisor, Dr. Nur Zincir-Heywood for her immense support and guidance during my Master's Degree journey. Apart from that, I show my deep appreciation to Global Affairs Canada for administering "Study in Canada Scholarship" for my studies at Dalhousie University.

This thesis is the result of hard work and commitment coming not only from my side but from my parents who have supported and encouraged me to study, develop and grow.

Chapter 1

Introduction

The role of Domain Name System (DNS) protocol is crucial to the core functionality of the Internet. DNS is the Internet's hierarchical, distributed database system that translates the Internet Protocol (IP) addresses to Domain names, and vice versa. Recently, this critical network functionality has been facing pressure for change. Specifically, two aspects are of particular relevance: DNS over HTTPS (DoH) and DNS over TLS (DoT). The objective is to secure communications between the DNS resolver residing on the end user's system and their chosen recursive resolver. DoH achieves this by embedding DNS query into a Secure HyperText Transfer Protocol (HTTPS) request/response, while DoT achieves this over the Transport Layer Security (TLS over TCP) protocol [73, 64, 35]. DoT protocol uses new TCP port number 853, and DoH blends into HTTPS traffic, sharing the same port. For this reason, controlling DoT traffic seems doable for network security specialists who can simply monitor and block malicious traffic going through the DoT-used port. Applying the same approach for DoH traffic is not possible [37]. New protocols potentially imply that significant changes will result in web and domain name mapping services. These new versions will change connection management layer, as well as changing the nature of traffic and application behaviours. Moreover, with DoH, the entire Internet 'threads' through the 'eye' of HTTPS. As a consequence, traditional ways of analyzing network and application data for cyber security, network operations and management using metadata, port or payload will no longer be possible [68, 63, 59].

In particular, DoH has been advocated to provide user privacy and security by encrypting the data between the DoH client and the DoH-based DNS resolver. As a consequence, it is argued that the risk of DNS data manipulation substantially decreases. What seemed to be a quite promising and effective solution at the very start, later gained a lot of criticism among the researchers, who claimed that DoH makes DNS tunnels harder to detect and mitigate. This specifically affects the detection

of malicious behaviours that exfiltrate data through DNS tunnelling. DNS protocol, which works with plain text for its data transmission, allows organizations to monitor DNS traffic by observing DNS queries. Once the data gets encrypted, threat analysis based on the plain text content of DNS queries becomes an obsolete tool for network security specialists. The features extractable from DNS queries, like domain name, record type, unique query ratio, query volume and length would be encrypted in DoH leaving only IP address, port number and timestamp in plaintext form [37]. The feeling of worry builds up with an overall increase in encrypted traffic. According to statistics[20], 95% of web data across Google platforms undergoes HTTPS encryption: this includes Advertising, Google Calendar, Google Drive, Gmail, Google Maps and YouTube. In addition to it, Operating Systems like macOS, iOS and Windows have been supporting DoH and DoT protocols. Web-browsers, like Google Chrome, Microsoft Edge, Mozilla Firefox and Opera have been working with newly established protocols, which can be configured from a settings panel [70]. Given that a form of encryption is applied in this newly established protocol, attackers can leverage DoH for malicious purposes. It is well known that DNS and its plaintext nature remains highly vulnerable to amplification attacks, DNS cache poisoning, botnet attacks, phishing attacks and DNS manipulation [58]. Due to these security issues, it is important to detect malicious DoH traffic.

Taking all these factors into account, researchers have started to explore host-based and network-based monitoring for DoH protocol analysis [52]. To this end, some recent works have evaluated the use of Machine Learning (ML), entropy, and network packet distribution-based approaches for analyzing DNS tunnelling and exfiltration attacks [66, 40, 75, 57]. While some of these works focus on using DNS-specific attributes, others use traffic or malware-specific attributes. This thesis explores the effect of entropy of a network flow in order to detect malicious behaviours in DoH tunnels. Even though previous works [47, 54] have employed entropy for summarizing network packet distributions, to the best of my knowledge, this is the first work studying entropy in the context of network flows. Additionally, the usability of AutoML tools, namely TPOT, for optimizing the DoH tunnelling classification is also explored. TPOT is a Python-based ML tool that was developed in 2016 with a core objective of optimizing ML pipelines using Genetic Programming [11]. Considering

thousands of pipeline configurations based on the dataset provided, TPOT automates the most tedious and time-consuming part of ML. Built on top of the scikit-learn library, TPOT currently has an implementation of 11 classifiers and 12 pre-processors required for training purposes [29]. Previous research deployed another AutoML tool called AutoAI for detecting malicious DNS over HTTPS traffic [31]. Based on the current literature studied, using the TPOT-AutoML toolset for malicious DNS traffic detection is the first experiment in its nature being reported. Thus, the novelty and the new contributions of this thesis are summarized as the following:

- Exploring the use of flow entropy characteristic to augment statistical features of network traffic flows for identifying malicious DNS, in particular DoH, tunnels;
- Exploring the minimum number of packets required to calculate entropy per traffic flow without decreasing performance or increasing complexity of the identification of malicious DoH tunnels;
- Exploring the optimization of the proposed approach in terms of complexity and performance to detect malicious DoH tunnelling behaviours using the TPOT-AutoML tool.

The rest of the thesis is organized as follows. Chapter 2 summarizes the related literature. Chapter 3 introduces the proposed approach and discusses the methodology used in this research. Chapter 4 details the experiments performed and presents the results obtained. Finally, conclusions are drawn and the future research is discussed in Chapter 5.

Chapter 2

Literature Review

Detection of attacks by analyzing network traffic has become one of the most widely researched areas in the cybersecurity world. Scientists have been proposing state-of-the-art Intrusion Detection Systems (IDS) and carrying out experiments to analyze malicious activities. In this thesis, the focus is on malicious DNS (DoH) tunnelling behaviours. Thus, this chapter is divided into four sections, with Section 2.1 summarizing DNS tunnelling and exfiltration detection research in general, Section 2.2 looking at DoH tunnelling identification in particular, Section 2.3 highlighting scientific work related to entropy usage for attack detection and lastly, Section 2.4 studying papers related to TPOT-AutoML tool and its effectiveness.

2.1 DNS tunnelling and exfiltration detection

I will start by looking at papers that studied DNS exfiltration and tunnelling detection methods. In [41], Das et.al. proposed ML algorithms to detect DNS channel exploitation, a possible alternative for traditional detection mechanisms like blacklists or signature-based methods. An application of a Machine Learning based system was designed for internal network enterprise. They came up with the first end-to-end system that identifies exfiltration and tunnelling activity based on internal packet data. For the dataset, researchers collected network traffic of a certain enterprise for 39 days, containing no malicious activity. For exfiltration data, they had to synthesize it artificially, with three sets using base64, base32 and hex on random strings and one set using base64 on credit card numbers. Once the dataset was generated and labelled, researchers calculated features like normalized entropy of concatenated string, length of the concatenated string, the ratio of uppercase and lowercase letters in the string, etc. Feeding 8 features into Logistic Regression model, authors demonstrated F1-measure of 96%, low false-positive and high detection rate.

In [54], Ahmed et.al. set the same goal of detecting DNS exfiltration and tunnelling behaviour from enterprise networks by performing real-time analysis of DNS queries. They collected DNS traffic from two enterprise networks, a medium-sized research institute and a large University campus, injected a million malicious DNS queries and fed a set of features extracted from queries into the Isolation Forest algorithm. To avoid the high computational cost of deriving time-series features of DNS queries, authors identified a set of three attributes: character count, the entropy of a query string and length of discrete labels in the query name that primarily helped to distinguish malicious DNS queries from benign ones. Performing fine-tuning during a training phase, they achieved 95% accuracy for malicious and 98% accuracy for benign classes for testing datasets. Since detection of DNS exfiltration and tunnelling was done in real-time, the authors also presented the average time complexity of the proposed methodology, demonstrating that 800 µsec would be enough per each query name.

Another work to identify DNS tunnelling and exfiltration activity was carried in [40], where Campbell et. al. used Self-Organizing Maps. Packet inspection-based approach was the core path chosen by the authors. Researchers represented DNS packets as query strings and extracted features from them, which were then used to train a Self-Organizing Feature Map (SOM), an unsupervised learning algorithm, to cluster DNS tunnelling, exfiltration and normal behaviours. Three publicly available datasets were used for representing benign behaviour, while publicly available tools like DNScat2, DET, DNSTunnel and DNSteal were run to generate tunnelling and exfiltration attack behaviours. In order to detect DNS exfiltration, authors extracted 8 features from query strings, including their length, ratio and etc. As for identifying DNS tunnelling, researchers derived 10 features. Derived features were put forward to prove the author's hypothesis: testing SOM's ability to self-organize the input data into distinct neighbourhoods, with neighbourhoods representing different behaviours (benign and malicious). Once the experiments were done, researchers achieved an F-measure of over 99% on all testing schemes, demonstrating the robustness of their model. SOM model was able to cluster all the data sets employed with high accuracy, separating malicious behaviours from benign ones with high precision.

Lastly, in [61] Lambion et. al. deployed Random Forest and Convolutional Neural

Network (CNN) to detect DNS tunnelling in real-time. Researchers created a dataset consisting of real-time DNS data, collected from subscribers including Internet Service Providers, schools and businesses. For each instance in a dataset, 5-tuple information was gathered: ⟨query name, query type, IP address, query time and date⟩. A label for each instance was provided as well: "non-tunnelling traffic", "normal resolved", and "DNS tunnel". Because only a limited amount of data can be tunnelled within each query name, data exfiltration is done over multiple queries combined. Hence, the authors of the paper grouped queries with the same SLD, day and IP address. CNN classifier was instructed to classify each instance of the dataset and subsequently use majority voting to label a group of DNS queries, while RF classifier was trained to classify entire groups of DNS queries. RF algorithm used 100 trees for training purposes and 11 features were fed into it. Results of classification demonstrated 96.04% accuracy for Random Forest and 99.30% for CNN model.

2.2 DoH tunnelling detection

Over the last decades, Network Security specialists have been raising a question about the security problem of DNS protocol. As a solution to this, DNS over TLS and DNS over HTTPS protocols have been suggested. DoT is an Internet Engineering Task Force (IETF) standard deploying TCP as its connection protocol to layer over TLS encryption and authentication between a DNS client and a DNS server. Functioning at the operating system level, it communicates over TCP port 853. Whereas, DoH leverages HTTPS for encryption and authentication between a DNS client and server. DoH shares TCP port 443 with HTTPS traffic, and unlike DoT, it is implemented at the application layer, creating room for browser traffic to bypass enterprise DNS control. As it was mentioned earlier, analyzing and controlling DoT traffic is still possible for Network Administrators who can monitor it as DoT traffic appears. Doing the same thing for DoH is not possible, since DoH shares a port with other HTTPS traffic. For this reason, focusing on DoH tunnelling detection has been set as a key priority for this research [26].

Among the latest research working with DoH tunnelling detection is the paper published by a group of scientists from the University of New Brunswick. In [66], Montazeri et. al. presented a two-layered approach to detect and characterize DoH

traffic using time-series classifiers. In the first layer, traffic was classified into DoH and Non-DoH, while in the second layer characterization of DoH into benign and malicious took place. They extracted 28 statistical features of DoH flows, using a tool called DoHMeter [4]. Statistical features included parameters like number and rate of flow bytes sent, number and rate of flow bytes received, packet length, packet time, etc. The authors employed six different machine learning (ML) classifiers at layer-1 and layer-2 and achieved an F-measure of 99.3% with Random Forest (RF) and Decision Tree (DT) classifiers. For their research experiments, authors generated benign and malicious DoH as well as Non-DoH traffic within their network premises, identifying every flow of encrypted network traffic by using tuple ⟨source IP, destination IP, source port, destination port, protocol ⟩. The dataset was made publicly available at [2].

Following this, in [75], Singh et. al. applied several Machine Learning algorithms to detect malicious activity in DoH and traditional DNS traffic. Researchers raised the question of DoH security risks since the new protocol bypasses local security measures like Firewalls, IDS and makes the audit of traffic impossible. The authors used the DoHMeter tool to extract statistical features from the publicly available DoH dataset [2] and employed five ML classifiers. It should be noted here that they applied the same 28 features as in [66]. Similar to previous research, the Random Forest algorithm outperformed others with Precision, Recall and F-measure reaching 99.99%.

Analyzing research works done in DoH tunnelling detection, a recent paper [32] published in August 2021 was studied. Behnke et. al. compared the performance of ten ML classifiers using ten-fold cross-validation on DoHBrwDataset [2]. The authors also used DoHMeter [4] tool to extract statistical and time-series features and applied Chi-Square and Pearson Correlation Coefficient (PCC) tests to address the overfitting problem. As a result, out of 34 statistical features, 21 were used for a classification task. The results showed almost 0% misclassification error for Light Gradient Boosting Machine (LGBM), Random Forest, Decision Tree and Extreme Gradient Boosting Machine (XGBM) classifiers. Apart from comparing performance metrics of ML classifiers, researchers also considered training and prediction time. Thus, LGBM was the fastest model, with training time making up 87 seconds for

Layer 1, and on the other hand, Random Forest was the slowest one, with training time reaching 1.39 hrs.

Lastly, a recently published paper [70] discussed detection of DNS over HTTPS tunnelling by using a set of ML algorithms. Researchers deployed a packet sniffer on their local PC to capture the packets of every connection layer transfer sent and received by the PC. Once the dataset was collected, they deployed Scapy - a module written in Python to manipulate data packets by exploiting numerous protocols. Python script, which leverages Scapy to clump packets into flows, extracted 34 features for each flow (it is interesting to note that features extracted by Scapy-based script were similar to features extracted by DoHlyzer tool in [66, 3]). To reduce the training time of the classification model, authors applied Gini index [19] to identify features to remove from the huge dataset. A set of Machine Learning algorithms used during the experiments were Logistic Regression, KNN, SVM (Linear and RBF), and Random Forest. Accuracy for the following classifiers ranged from 94% to 99%, where RF outperformed others with 99.8% accuracy. The training time of the model ranged between 2.2 seconds and 39.7 seconds, with LR being the fastest model.

2.3 Using entropy

In the literature, cyber security specialists have studied the use of entropy features to detect malicious behaviours in DNS and other network traffic. In [65], Mejri et.al. used packet entropy to identify Denial Of Service (DOS) attacks happening at Vehicular Ad hoc Networks (VANETs). Packets, circulating within the network, were categorized into four types: DATA, ACK, RTS and CTS packet families. After that, the authors calculated the entropy value of DATA and ACK packets. Comparing entropy value results of different scenarios, researchers managed to prove their assumptions: higher entropy of packets corresponded to normal VANET behaviour. The gap of entropy values allowed the proposed schema to distinguish easily between benign and malicious VANET activity.

The paper[40] mentioned in the DNS tunnelling detection section used the entropy feature as well. The packet inspection-based approach was followed throughout the experiments. The authors extracted eight features from DNS queries, including normalized entropy of concatenated string to detect DNS exfiltration. A similar

method was undertaken for DNS tunnelling detection, where entropy was among the ten features derived from DNS queries.

Another work [30] looked at the entropy value of NAN characters present at Uniform Resource Locator (URL) strings to apply at URL-based phishing detection. The core purpose behind computing entropy was to look at how NAN characters were distributed in each URL. Augmenting entropy value with previously proposed 10 features, like IP address, age of the domain, port number, etc, researchers fed 11 features into Random Forest classifier. As for datasets, authors made up balanced and imbalanced datasets. The balanced dataset consisted of an equal number of legitimate and phishing URLs, while the ratio of benign and malicious URLs in the imbalanced dataset was 9:1. Once the experiments were completed, they compared the efficiency of the entropy feature and concluded that augmenting entropy with other features increased the model's precision rate by 7-10%. Despite significant improvements in accuracy when applying entropy, authors undermined the high False Positive Rates still present in both cases.

In [79], Zhou et. al. worked with entropy-related features to detect spam emails using gcForest - deep forest learning algorithm. Authors calculated three features for spam detection: entropy of subject size of sent mails, the entropy of content size of sent emails and the ratio of received mail count to sent mail count. In terms of the dataset, they collected mail data at a border network of a province in China for 34-days. Once the dataset was collected, they ran proposed features on three different classifiers: ALAC, SVM and gcForest. Results showed high spammer detection rates for all three classifiers at the source network, exceeding 90%. Researchers managed to demonstrate the efficiency of entropy-related features for identifying spam mails.

Similarly in [47], Fawcett designed the ExFILD tool to detect data exfiltration by analyzing encryption characteristics of the packets and sessions, where features extracted from network traffic were classified by a Decision Tree. Fawcett processed network traffic at the packet and session levels. For extracting packets belonging to one session, the researcher used a session identifier consisting of source and destination IP addresses and their respective ports. Entropy values of packets' payload were calculated to model the encryption state for packets or sessions. Then, they were input to the Decision Tree to classify whether a packet/session contains exfiltrated

data. The decision-making tree was a key body in the proposed methodology to decide whether a packet or a session contains exfiltrated data. The tree branched out into four branches: expected and received unencrypted data, expected unencrypted but received encrypted traffic, expected encrypted but received unencrypted data, and lastly, expected and received encrypted data. Once encryption characteristics of a packet/session went through the tree, it decided whether or not to flag a specific packet/session.

2.4 Optimization using TPOT-AutoML

In [67] Olson et.al. implemented an open-source TPOT tool using Python library and presented the tool's efficiency by testing it on a series of simulated and real-world benchmark datasets. The main reason behind developing TPOT was to make machine learning a more accessible, scalable and flexible area for the research community. Pipeline operators implemented in TPOT were based on the scikit-learn module: preprocessors, decomposition, feature selection and models. Researchers primarily focused on supervised learning models in the last operator, like Decision Tree, Random Forest and Gradient Boosting (GB) classifiers, along with SVM and Logistic Regression models. To demonstrate TPOT's efficiency, they picked nine supervised learning datasets from the well-known UC-Irvine Machine Learning Repository [44]. They divided all datasets in ratio 3:1, ran the TPOT tool and concluded that experiments with large datasets demonstrated a higher accuracy rate. The authors also underlined an issue with TPOT's slow activity on a large dataset, which could take several hours to complete the training process.

Another paper [48] compared eight open-source AutoML tools benchmarking them on 12 popular OpenML datasets. Ferreira et. al. carried out experiments based on three scenarios: General Machine Learning (GML), Deep Learning (DL) and XGBoost. They divided datasets into ten folds to perform external cross-validation. In addition, they split datasets into a 3:1 ratio, allocating 75% of each for training. Researchers used different performance metrics for regression, binary and multi-class classification tasks. Mean Absolute Error was used for regression, Area Under Curve (AUC) for binary classification and Macro F1 score for multi-class classification. In

total, authors performed 12 (datasets) x 6 (tools) x 10 (folds) = 720 AutoML executions. In the GML scenario, TPOT was the top second tool in terms of execution time, which explains its long training time. For test datasets, TPOT achieved a 74% F1 score for multi-task classification, performing better than the other 4 AutoML tools. In other scenarios, TPOT did not manage to set high scores.

The next paper [31] used another AutoML tool, called Auto AI for malicious DoH traffic classification. Banadaki et.al. deployed DoHBrw dataset [2], extracted 34 features by using the DoHlyzer tool [3] and fed them into Auto AI to analyze different pipeline configurations the tool provided. Unlike TPOT, Auto AI considered six Machine Learning classifiers, namely Decision Tree classifier, Extra Trees classifier, Gradient Boosting classifier, LGBM and XGB classifier, and lastly Random Forest classifier. Results demonstrated the high performance of LGBM and XGBoost classifiers, achieving an accuracy of 100% in separating traffic into DoH and non-DoH. As for identifying malicious DoH among benign DoH traffic, LGBM outperformed the other five classifiers again.

Last, but not least, paper [42] discussed the application of the ANTE system, AutoML tool for botnet detection among IoT devices. Neira et. al. proposed the system that autonomously selected the most appropriate ML pipeline for different botnet attack scenarios. ANTE tool was based on the Auto-Sklearn framework, consisting of 3 parts: data collection and preprocessing, feature processing and estimator. AutoML tool had 14 options for pre-processing features, 15 supervised classifiers implemented. Researchers deployed three different datasets to test the proposed methodology, extracted 20 features from network traffic and provided them as an input to the ANTE system. The system, in its turn, identified the best model, executed a bot anticipation process and notified the network security administrator about malicious behaviours. The work demonstrated an average accuracy of 99.87% for botnet detection.

2.5 Summary

As discussed above, the field of DoH Tunnelling detection has been a popular topic over recent years. Extraction of specific features from DNS traffic packets and flows as well as analysis of the use of entropy on normal and malicious packets - are examples of work carried out recently. Researchers have been suggesting novel approaches to

increase detection rates of malicious activities in a new DoH protocol. The encrypted nature of DoH traffic complicates the work for cyber security experts, pushing the need for comprehensive, fast, and yet reliable analysis models.

To the best of my knowledge, no research has explored the use of entropy on network traffic flows in terms of modelling different behaviours such as tunnelling and exfiltration in encrypted DoH tunnels. Moreover, this is the first time such an approach gets optimized using the TPOT-AutoML system. The potential of TPOT to find the best parameters and model ensembles by using a genetic search algorithm is promising and encouraging. In other words, TPOT enables the identification of a pipeline configuration that might not get considered otherwise, bringing in a new perspective by a systematic search of the solution space.

To sum up, the research papers discussed above studied and proposed a wide scope of DNS and DoH tunnelling detection systems. While some of the papers carried deep-packet inspection and performed computationally expensive packet analysis [41], which may complicate real-time malware detection, other works utilized network flow feature extraction tools, which dealt with TCP protocol only [66, 75, 32]. Hence, the approach these papers suggested is not generalizable to the extent required by research and the cyber security community.

Studies related to entropy utilization for attack detection calculated it at a packet level, resulting in a time-consuming and computationally expensive approach [65, 30, 79]. On top of that, some experiments were done by extracting query strings from DNS requests, which may not work for DNS over HTTPS protocol, since DNS requests get encrypted by an increasing number of DoH-based web browsers.

Chapter 3

Methodology

In this thesis, the proposed approach for detecting tunnelling and exfiltration behaviours in DoH traffic is a network traffic flow inspection-based approach, where the flow features are augmented with the entropy of the network flow. This augmented feature set is then used with ML classifiers to detect the malicious DNS tunnels. Figure 3.1 shows the overall methodology followed in this research. Three publicly available datasets are run through Flow Exporters, generating statistical features for each flow, which are then augmented with the entropy of a flow. For the entropy feature, four different scenarios are benchmarked during the experiments: the first one does not deploy any entropy feature, the second one calculates an entropy value per flow by using all packets of the given flow, the third one calculates an entropy value per flow by using the first 96 bytes of a packet's payload and the last one calculates the entropy value using only the first n-number of packets of a flow (where n=4-6). The final set of statistical and entropy features are input into ML classifiers, including the C4.5 Decision Tree, Random Forest, Logistic Regression (LR), Support Vector Machine (SVM) and Naive Bayes (NB). After the model is trained, five different testing datasets are used to evaluate the performance, complexity and computational cost of the trained models to differentiate benign flows from malicious ones.

Once the final statistical features are chosen along with the entropy of a flow, the proposed approach is optimized further using the TPOT-AutoML system. It should be noted here that TPOT is trained on the same training dataset, and then five different testing scenarios are analyzed for optimizing the performance of the proposed detector approach. The reduced number of features required for training ensures considerably less time spent on finding the best possible pipeline by the TPOT-AutoML system.

This chapter is structured as follows: Section 3.1 presents the datasets used for this research, Section 3.2 provides information about network flow extraction tools

used, Section 3.3 introduces the proposed approach of augmenting flow statistical features with the flow entropy, Section 3.4 provides information about Weka tool used throughout this research, Section 3.5 discusses deployed algorithms via Weka, and finally Section 3.6 discusses the TPOT tool and explains the experiments completed using TPOT for optimization of the proposed approach.

3.1 Datasets

Due to the novelty of the DoH protocol, finding publicly available datasets is a challenging task. Montazeri et. al. released the "CIRA-CIC-DoHBrw-2020" dataset [2], where they generated Benign-DoH, Malicious-DoH and non-DoH traffic using five different browsers and four servers. Researchers used Google Chrome, Mozilla Firefox browsers along with dns2tcp, DNSCat2 and Iodine tools to access the top 10k Alexa websites. As for servers: AdGuard, Cloudflare, GoogleDNS and Quad9 were employed to respond to DoH requests. In this dataset, DoH traffic is divided into benign and malicious, with malicious activity being represented in a tunnelled form. Tools used for malicious traffic generation sent TCP traffic encapsulated in DNS queries. DNS queries, in its term, were sent using TLS-encrypted HTTPS requests to one of four DoH servers. Hereafter, this dataset is referred to as DoHBrw.

Another publicly available dataset used in this research is "GT Malware Passive DNS Data Daily Feed" [77, 5]. The dataset collection was initiated by Georgia Tech Information Security Center in 2015, with data capture continuing. It was generated by running suspect Windows executable files in a sterile and isolated environment, with limited access to the Internet. For this research, DNS Data for the year 2020 was deployed. Hereafter, this dataset is referred to as ImpactGT.

Additionally, one more publicly available dataset, namely "CICIDS 2017" [53, 74] is also employed in this thesis. Intrusion Detection Evaluation Dataset was released by the University of New Brunswick in 2017 and contains benign and malicious pcap files, represented in a raw form. Researchers attempted to simulate abstract behaviours of 25 users using HyperText Transfer Protocol (HTTP), HTTPS, File Transfer Protocol (FTP), Secure Shell (SSH) and email protocols for benign traffic. The capture of network traffic lasted for 5 days, with Monday representing benign activity and the rest of the four days focusing on malicious behaviours. I made use of only benign

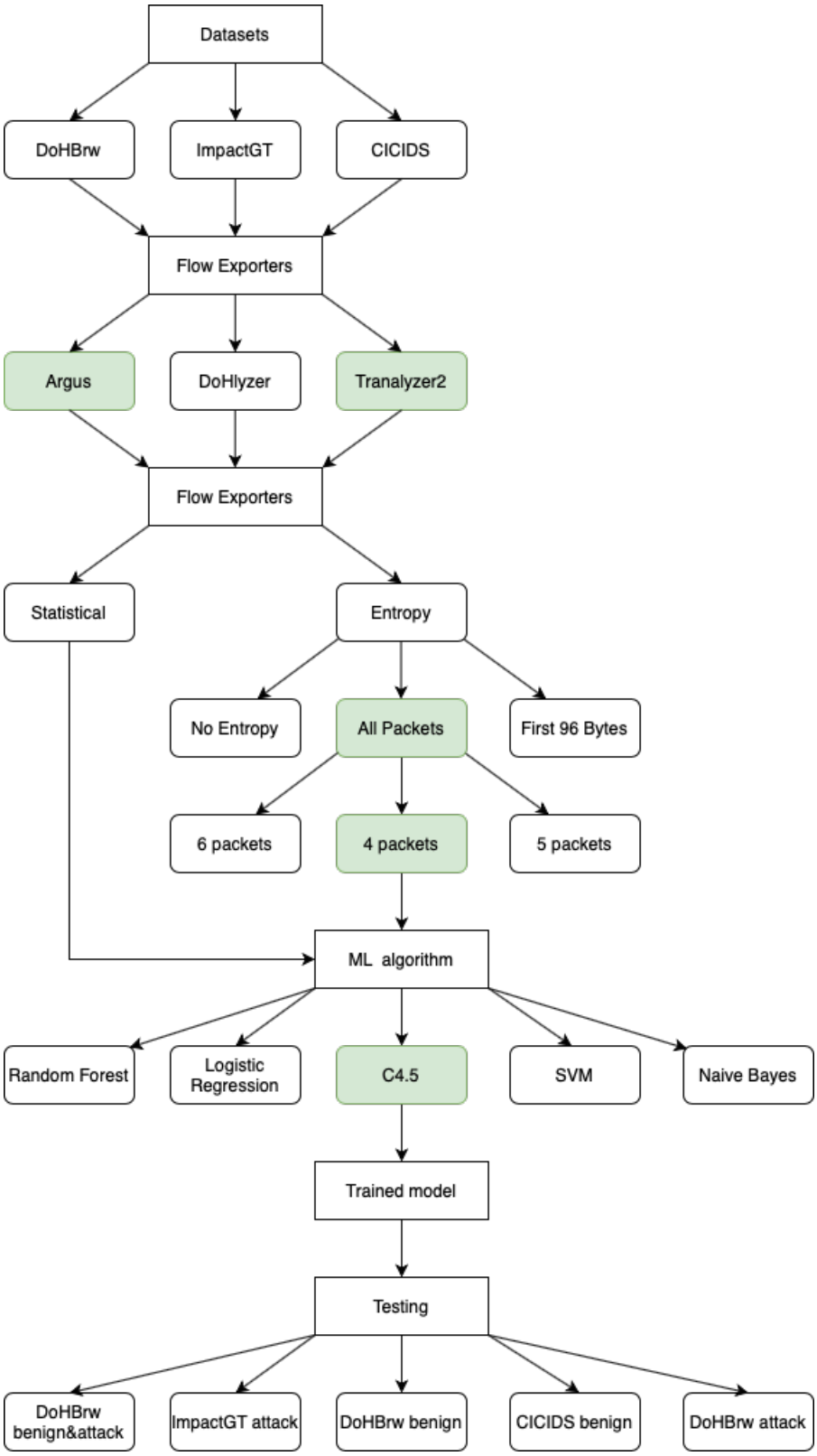

Figure 3.1: Research Methodology for selecting the best feature set

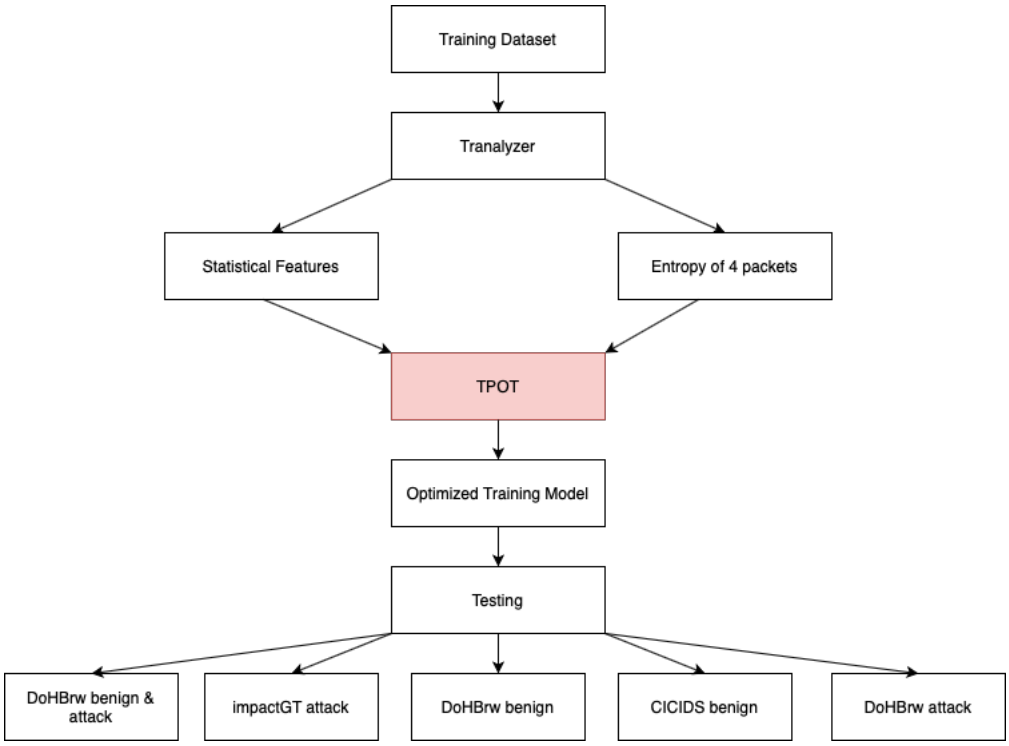

Figure 3.2: Research Methodology for optimizing classification using the selected best feature set

traffic from this dataset. Henceforward, it is referred to as CICIDS.

Table 3.1 shows the number of flows used from each dataset in my research. As it can be seen from the table, the largest number of flows were taken from the DoHBrw dataset, corresponding to benign and attack behaviour. ImpactGT dataset was only represented in malicious form. On the other hand, CICIDS flows outlined benign user activity. The datasets used in this thesis are balanced with an equal amount of malicious and benign flows for training and testing purposes. Table 5.4 in the Appendix part, provides information about the types of attacks represented in each dataset, and the way benign traffic was generated.

Datasets	DoHBrw	ImpactGT	CICIDS
Number of flows	33000	12000	12000

Table 3.1: A summary of the datasets used

3.2 Network Flow Extraction Tools

As discussed earlier, in this research, I explore the application of network traffic flow-based features augmented with the flow entropy calculated. It enables the analysis of encrypted DoH traffic for malicious behaviours since no deep packet inspection is necessary. Hands down, when it comes to anomaly detection, packet-based inspection analysis ends up being a costly solution looking at packet payload and header on an individual basis. By contrast, a flow-based approach considers flow properties in general - duration, number of bytes sent and received (among others) - making real-time analysis of network activity possible for network security specialists. To this end, I explore the usage of the following three network traffic flow exporters that are publicly available.

1. *Tranalyzer2* (T2) is a lightweight flow generator and packet analyzer, which can work with ultra-large packet dumps. Tranalyzer2 consists of a core and a set of plug-ins, which users can activate according to their needs. Packet-to-flow aggregation provides better analysis of network operations [38]. Similar to TShark, Tranalyzer2 supports packet mode, but unlike TShark, Tranalyzer2 also includes a unique numerical ID linking every packet to its flow [39]. In [50], Haddadi et. al. compared several flow exporter tools and reported that Tranalyzer2 [6] demonstrated the best performance. The functionalities of Tranalyzer2 are as the following:

- Packet capture
- Packet-to-flow allocation
- Flow timeout handling
- Plug-in function invocation
- Flow/packet based output formats

Unlike other flow exporters, Tranalyzer2 provides information about flow direction, labeling A and B flows (client to server and server to client respectively). Overall, thanks to the large number of statistical features extracted (Tranalyzer2 extracts 109 features for each network flow), T2 allows analysts perform troubleshooting and provide network and application security. On top of that, Tranalyzer2 has become

preferred tool for researchers, who use it for traffic preprocessing before training their ML classifiers for malicious traffic detection.

2. *Argus* is a bi-directional network traffic flow monitoring system. It provides information about network flow status and is generally used for Network Intrusion Detection and Anomaly Detection projects [1]. Released in 1993 and implemented in C language, Argus was used for network monitoring, supporting distributed network architecture [78]. Argus supports many protocols, like TCP, ARP, ICMP, ESP, using its binary format for flow extraction. For this reason, it is required to use the ra tool to convert the binary output into CSV. Ra module of the tool reads the input file specified by the user and writes user-specified fields to a file. The maximum number of fields Argus can extract is 125. For this research, I decided to extract all flow features and analyze how ML classifiers would use them for classification purposes.

3. *DoHlyzer* is specifically designed to export and analyze DoH traffic flows by researchers from the University of New Brunswick. Developed in Python, DoHlyzer reads user-specified PCAP files and extracts statistical and time-series features into CSV files. DoHlyzer extracts 34 features for each flow, including the number and rate of flow bytes sent/received, mean and median packet time, the standard deviation of packet time, and so on. DoHlyzer consists of several modules that assist data analysis of DoH flows. These include DoHMeter, Analyzer and Visualizer [3]. Functionalities of the DoHMeter model are:

- Capturing HTTPS packets from network interfaces and parsing user-specified PCAP files;
- Grouping packets into flows by their source and destination IP addresses as well as source and destination port numbers;
- Statistical and time-series feature extraction for traffic analysis.

To compare these three flow exporters, I employed a C4.5 decision tree to classify attack versus normal behaviours using the training dataset. In this set of experiments, I only used the statistical flow features without augmenting them with the entropy, Table 3.2. The results showed that while Argus and DoHlyzer achieve F-measure values around 99%, Tranalyzer2 achieves F-measure of 95%. Since all of them have high performances, I further analyzed the features that were selected by the trained

decision tree model from the set of all features. To this end, I observe that the C4.5 classifier selected seven features to separate normal flows from malicious ones while using Tranalyzer2. It selected ten features while using Argus, and fourteen features while using DoHlyzer. Another potential downside for DoHlyzer is that it can only support feature extraction based on TCP flows, whereas Argus and Tranalyzer2 work with TCP and UDP flow.

Flow Exporter	# of attributes used	P	F	R
Tranalyzer2	7	95.6	95.2	95.2
Argus	10	99.6	99.6	99.6
DoHlyzer	14	99.9	99.9	99.9

Table 3.2: Performance of the C4.5 training model on flow statistical features without entropy by three flow exporters

Given the low number of features selected when Tranalyzer2 is used (less number of features potentially enables a simpler ML model and near real-time execution) and the ease of using Tranalyzer2 (compared to DoHlyzer and Argus not working robustly) on all different datasets employed, I selected to continue with Tranalyzer2 to extract features from flows for the initial part of experiments. However, I did compare Tranalyzer2-extracted features to Argus-extracted flow features, given that both tools support TCP and UDP. These comparisons are included in the Results section of the thesis, demonstrating the better performance obtained by Tranalyzer2-extracted features.

3.3 Network Flow Entropy and Statistical Features

Claude E. Shannon introduced the concept of entropy for information theory in 1948 [46]. Entropy is a measure of a state of randomness, disorder or uncertainty. The more random the string is, the higher the entropy value it has. Contrary, a string of all same-character letters will produce an entropy value of 0. Entropy has been used in many fields from thermodynamics to physics and has been of interest in network anomaly detection as well. Detection of online warms, DDoS attacks, ransom malware - are just a few examples where entropy can be used for attack identification purposes. Entropy can be considered a handy approach compared to traditional attack detection tools due to its need for a small number of packets for entropy value calculation. We all

#	Attribute	Description
1	tcpSeqSntBytes	TCP sent seq diff bytes
2	connSip	Number of connections from source IP to different hosts
3	tcpSSASSAATrip	(A) TCP Trip Time Syn, Syn-Ack;(B) TCP Trip Time Syn-Ack
4	tcpRTTAckTripJitAve	TCP ACK trip jitter average
5	aveIAT	Average inter-arrival time
6	bytps	Sent bytes per second
7	stdIAT	Standard inter-arrival time

Table 3.3: Features chosen by Tranalyzer2

#	Attribute	Description
1	DstLoad	Destination bits per second
2	TcpRTT	TCP connection setup round-trip time, the sum of 'synack' and 'ackdat'
3	Cause	Argus record cause code: Start, Status, Stop, Close, Error
4	Dur	Record total duration
5	Dir	Direction of transaction
6	sTtl	Source to Destination TTL value
7	Rank	Ordinal value of this output flow record i.e. sequence number
8	DstLoad	Destination bits per second
9	Proto	Transaction Protocol
10	dHops	Estimate number of IP hops from dst to this point

Table 3.4: Features chosen by Argus

#	Attribute	Description
1	FlowBytesSent	Number of flow bytes sent
2	PacketLengthMode	Mode packet length
3	DestinationPort	Destination port
4	PacketLengthMedian	Median packet length
5	PacketTimeMedian	Median packet time
6	PacketLengthSkewFromMedian	Skew from median packet length
7	ResponseTimeTimeVariance	Variance of request/response time difference
8	FlowReceivedRate	Rate of flow bytes received
9	PacketTimeSkewFromMedian	Skew from median packet time
10	PacketLengthVariance	Variance of packet length
11	PacketLengthSkewFromMode	Skew from mode packet length
12	Duration	Duration
13	ResponseTimeTimeMode	Mode request/response time difference
14	PacketLengthMode	Mode packet length

Table 3.5: Features chosen by DoHlyzer

know that traditional IDS requires a huge volume of data to analyze traffic behaviour, and quite often hidden malware activities remain undetected by standard tools.

To this end, researchers have used entropy to capture important characteristics of a packet's header or payload distributions [60, 33, 54, 40] for detecting anomalous behaviours. However, I am not aware of any work that provides a methodology to leverage entropy over network flows. Thus, the research hypothesis is that entropy characteristics of a network traffic flow could provide an accurate metric to show the randomness and therefore could be used to indicate the actual state of encryption in the flow analyzed. Campbell et. al. has shown that entropy values calculated over encrypted packet payloads enable identification of tunnelling behaviours in DNS traffic in [40]. Thus, to test my hypothesis I study the use of entropy over network traffic flows. The following equation is used to calculate the entropy of a network flow data:

$$H(X) = -\sum_{i=1}^{N} p(X_i) \log_2 p(X_i) \tag{3.1}$$

where X is the string and X_i is a character in the string. $p(Xi)$ is a particular character's probability of being present in the string [46, 47].

Once the datasets were chosen for my research, finding the right algorithm for flow entropy calculation was an essential step forward. Figure 3.3 shows the detailed approach I followed in network flow entropy calculation. The raw pcap file is provided as an input file for the T-Shark tool, which extracts 6-tuple information from each packet: ⟨source IP address, destination IP address, source port, destination port, packet's frame time, packet's payload⟩. T-Shark writes output into JSON file, which is later fed into MATLAB script from one side. The same PCAP file is provided as input for the network flow extractor. Tranalyzer2, for instance, outputs 106 flow features and the output is given as a CSV file. Then, this output is provided into MATLAB script from the other side. Once both JSON and CSV files are fed into MATLAB, the script runs to calculate the entropy of a flow.

JSON file, generated by the T-Shark tool, contains 6-tuple information, which MATLAB script uses to match each packet to a particular flow. The script matches source and destination IP addresses, source and destination port numbers, and packet frame time to construct the flows. After all of the packets are matched to their flows,

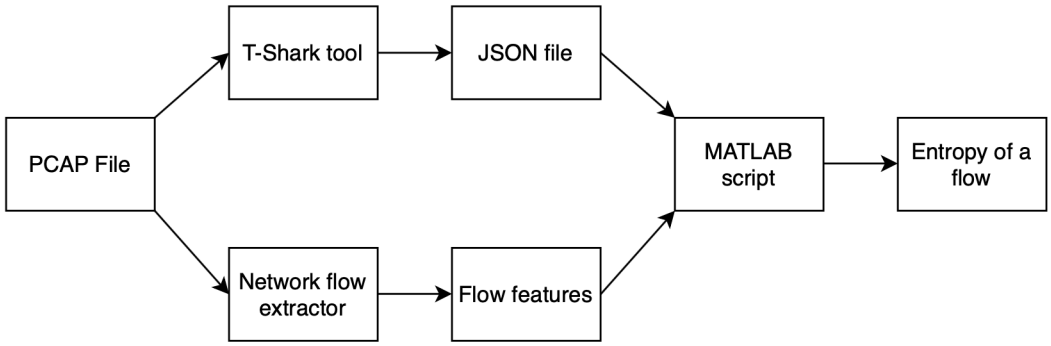

Figure 3.3: Methodology for entropy calculation

the entropy value of the merged n packets is calculated.

In this thesis, I augment statistical flow features with the entropy of a network flow. In this case, the decision tree employs 13 features in total, where 12 of them are statistical features selected by the decision tree from the set of 106 features that Tranalyzer2 extracts. It should be noted that features selected by the decision tree significantly differ from the features chosen when no entropy is employed (See Table 3.6).

#	Attribute	Description
1	%dir	Direction of the flow
2	numBytesSnt	Number of bytes sent
3	minPktSz	Minimum packet size
4	stdIAT	Standard inter-arrival time
5	ipMindIPID	IP minimum delta IP identification
6	ipMaxTTL	IP maximum time to live
7	tcpPSeqCnt	TCP packet sequence count
8	tcpInitWinSz	TCP initial effective window size
9	tcpAveWinSz	TCP average effective window size
10	tcpMSS	TCP maximum segment length
11	tcpWS	TCP window scale
12	tcpRTTAckTripMax	TCP acknowledgment trip maximum
13	entropy	Entropy value of a flow

Table 3.6: Features chosen by Tranalyzer2 augmented with entropy

Once the statistical and entropy features are extracted and calculated, a training dataset with 12000 malicious and benign flows is used to create a training model. To

ensure a balanced dataset, half of the 12000 flows are benign and half are malicious. In addition to it, the dataset is balanced in terms of protocols too: both benign and attack flows are represented with TCP and UDP protocols. DoHBrw dataset consists of TCP flows, while ImpactGT and CICIDS are made up of UDP flows.

As discussed earlier, the training model is evaluated under four different scenarios: (i) Only statistical flow features are used without any entropy, (ii) Flow features are augmented with the entropy that is calculated over all the packets of a network flow, (iii) Flow features are augmented with the entropy that is calculated over the first 96 bytes of a network flow [43], and (iv) Flow features are augmented with the entropy that is calculated for the first n-packets of a network flow. When scenario 2 outperformed scenarios 1 and 3, I explore the entropy concept of a network flow by using fewer data and therefore employ scenario 4. The reason behind this is that identifying all packets belonging to a flow, and calculating their entropy is computationally expensive. To decrease this computational cost, I analyze the number of packets per flow (following the work in [34]) in each dataset used, Figure 3.4. As it is noticeable from the figure, MATLAB script analyses 31 and 32 packages per single flow in DoHBrw testing datasets before merging their payload, 7.78 packages for training and 5.15 packets for ImpactGT datasets.

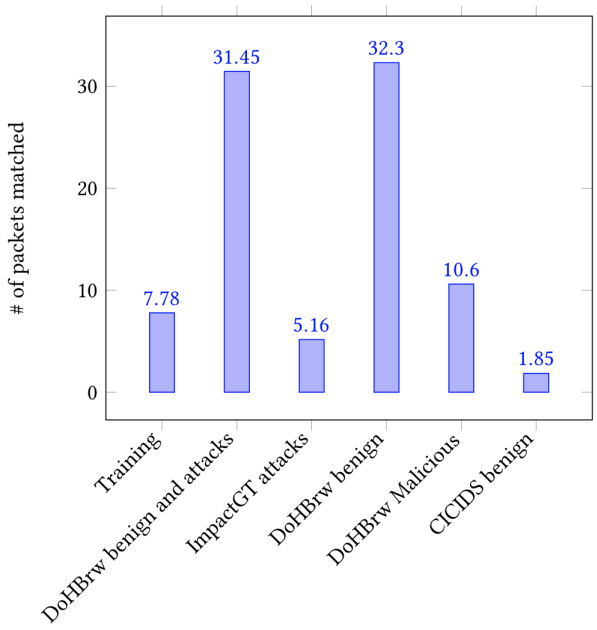

Figure 3.4: Average number of packets per flow in training and testing deployments

#	Attribute	Description
1	DstLoad	Destination bits per second
2	Entropy	Entropy value calculated over 4 packets
3	TotBytes	Total transaction bytes
4	DstBytes	Destination → Source transaction bytes
5	RunTime	Total active flow run time
6	sTtl	Source to destination TTL value
7	Rank	ordinal value of this output flow record i.e. sequence number

Table 3.7: Features chosen by Argus augmented with entropy

I, therefore, evaluate the performance of the trained classifiers by calculating the entropy for the first n=6, n=5 and n=4 packets of a given flow. In this case, my objective is to find the minimum number of packets per flow that can provide a reasonable indication of the entropy of a flow without decreasing the performance of the classifier. The results of these evaluations show that calculating flow entropy over the first four packets of a flow decreases the computational cost (relative to all packets) without decreasing the F-measure of a classifier. It should be noted here that when a flow has less than four packets, all packets belonging to that flow are used for the entropy calculation. The evaluation results are presented and discussed in more detail in the next chapter.

Having carried out experiments with Tranalyzer2 flow exporter and looking at how 13 statistical features help ML classifiers distinguish traffic behaviour, it was decided to try Argus flow exporter and analyze how Argus-extracted features help Weka tool classify the same flows in the same set of training and testing datasets. Table 3.7 shows 7 features deployed by the C4.5 decision tree for classification of the training model.

3.4 WEKA: Machine Learning Software in Java

WEKA, Waikato Environment for Knowledge Analysis, represents a collection of algorithms for data analysis and visualization tools along with Graphical User Interface (GUI) for convenient access to user[15, 7]. WEKA is a free software issued under the GNU General Public License. Developed at the University of New Zealand, WEKA was designed to help researchers carry Machine Learning experiments and apply developed techniques to real-world data mining problems. The workbench has integral

methods required for data mining problems: regression, classification, clustering, association rule mining, and attribute selection. Developers suggest three ways of using WEKA[45]:

- applying a learning method to a dataset and analyzing its output to learn more about data
- using learned/trained models to make predictions on new/unseen data
- applying various learners and comparing their performance to choose the best one for prediction

The data is usually presented in a spreadsheet or database. However, WEKA's native data storage format is ARFF. The ARFF file consists of a list of instances and the attribute values for each instance are separated by commas. The main difference between ARFF and CSV file is the presence of @relation, @attribute and @data tags in the first file format. @relation tag defines the name of the database, @attribute tag defines attributes, @data defines a list of data rows, in a comma-separated line. WEKA accepts numeric and nominal attribute values, so string attributes have to be removed from the dataset. Once the user loads the dataset into WEKA, he/she is able to choose the learning algorithm from a list of implemented ones. The current list of classifiers, for instance, includes but is not limited to Naive Bayes, Logistic Regression, J48 (C4.5), Random Forest, SVM and etc.

When choosing classifiers, a user is able to set parameters based on preference and training dataset. For J48, for instance, the user can play with the confidence threshold for pruning (default is 0.25), a minimum number of instances permissible at a leaf (default is 2), and others. Instead of standard C4.5 pruning, the user can opt for reduced-error pruning, instruct the classifier to apply Laplace for counts at a leaf or not.

After choosing a classifier, the next step is to choose the evaluation method that the tool offers: Cross-validation, Training set, Testing set and Percentage split. Unless the user has its own training and testing sets, choosing Cross-validation and Percentage split is recommended. For Cross-validation, a number of folds in which the entire data would be split and used at each iteration of training are set, while in

Percentage split, the user-specified ratio would be applied to data and divided into training and testing subsets.

As mentioned above, WEKA has powerful visualization tools, both for datasets themselves and for classification results. Dataset visualization tool displays a matrix of 2D scatter plots for each pair of attributes. The classification model visualization tool provides a visual representation of the trained Decision Tree model once the training process is completed (see Fig. 4.3). This function of the tool has been used a lot throughout the research to estimate tree's complexity when working with various network flow extractor tools. A user is able to see how many instances of the training set were classified correctly and incorrectly based on a particular attribute of the data.

3.5 Machine Learning Algorithms applied via WEKA

The main task of the Machine Learning classifier is to study data provided as an input and predict the output class for unseen instances. The simplest example of the classification task also referred to as binary classification, is spam detection, where the ML algorithm filters emails into "spam" and "not spam" categories [13]. In this research, Machine Learning classifiers are applied to classify network flows into benign and malicious, providing an efficient predictive model for network security specialists [36]. The key algorithm deployed in experiments is C4.5 Decision Tree, along with Random Forest, Logistic Regression, Support Vector Machine and Naive Bayes. This section talks about mentioned classifiers in detail.

C4.5 Decision Tree is a supervised Machine Learning algorithm, using a tree structure to solve a particular classification problem. It is an improved version of Quinlan's ID3 algorithm, which underwent a series of improvements[8]:

- C4.5 Decision Tree can work both with discrete and continuous attributes
- C4.5 can perform training process despite missing values
- C4.5 applies pruning once decision tree is created

In WEKA tool, J48 is an open-source Java implementation of C4.5 algorithm, allowing classification to be performed in two modes: through decision tree or rules generated

from them [56]. Default parameters of the C4.5 decision tree in WEKA can be changed by the user, but for my research, none of the parameters were changed. Parameters include pruning (whether to perform it or not), collapseTree (whether to remove parts of the tree that do not reduce training error), the minimum number of instances per leaf, useLaplace (whether counts at leaves are smoothed based on Laplace) and etc.

C4.5 builds up a decision tree based on the sample instances provided for training purposes. The algorithm is implemented using the concept of information entropy, the same concept utilized in its predecessor, ID3. Training set is represented as $S = \{s_1, s_2, ...\}$ with labeled instances of data, where each instance of data, s_i consists of p-dimensional vector $\{x_1, i, x_2, i, ..., x_p, i\}$ with x_j representing attribute value of corresponding instance as well as its class. The attribute with the greatest information is chosen to be split on by Decision Tree at each node, after performing normalized information gain calculation. C4.5 repeats this procedure on partitioned sub-lists following the divide-and-conquer approach. Finally, a decision tree is created based on a greedy algorithm. Once the training process comes to its end, C4.5 performs pruning in order to avoid overfitting problems. The algorithm leaves the largest generalizable part of the tree and prunes out the branches, contributing to better performance in testing instances[22]. C4.5 is known to use another concept of decision tree making process based on the Gini index. A detailed explanation of the splitting process is provided below.

Belonging to the family of decision tree-like algorithms, C4.5 undergoes recursive partitioning of the training set. The recursion continues until achieving subsets that are as pure as possible to a given target class. Each node of the tree corresponds to a specific set of records T identified by a particular test on a feature. For instance, continuous attribute A can be split by performing test $A \leq x$. The set of records T is then divided into the left and right branches.

$$T_l = \{t \, \epsilon \, T : t(A) \leq x\}$$

and

$$T_r = \{t \, \epsilon \, T : t(A) > x\}$$

Likewise, a set of categorical features B can be split into subsets based on its value. For example, support $B = \{b_1, ..., b_k\}$ each branch i can be induced by the test $B = b_i$.

The process of dividing features into splits of decision trees considers all possible combinations in search of the best one based on the splitting criteria. If a dataset is induced on the following scheme

$$A_1, A_2, ..., A_m, C$$

where A_j are attributes of the dataset and C is the target class, all candidates go through split generation and evaluation by the splitting criterion. The choice of the best split considers impurity measures for the decision-making process. The impurity of the parent node has to be decreased by the split. Let $(E_1, E_2, ..., E_k)$ be a split induced on the set of records E, a splitting criterion that uses impurity measure $I(\cdot)$ is:

$$\triangle = I(E) - \sum_{i=1}^{k} \frac{|E_i|}{|E|} I(E_i)$$

Out of two standard impurity measures, Decision Tree uses the Gini index, defined for the set E as following. Let p_j be the fraction of records in E of class c_j:

$$p_j = \frac{|\{t \, \epsilon \, E : t[C] = c_j\}|}{|E|}$$

then

$$Gini(E) = 1 - \sum_{j=1}^{Q} p_j^2 \tag{3.2}$$

where Q is the number of classes. It should be noted here that having records of the same class gives zero impurity [10].

Random Forest is a supervised learning algorithm, which can be used both for classification and regression tasks. It consists of decision trees, trained with the bagging (bootstrap aggregating) method. Given a training set $X = x_1, x_2, ..., x_n$ with responses $Y = y_1, y_2, ..., y_n$, bagging repeatedly selects a random sample with replacement of the training set and fits trees to these samples. Let us assume B to be the number of samples/trees, which can typically vary between a few hundred to several thousand [25].

For $b = 1, 2, ..., B$:

1. Sample, with replacement, n training examples from X, Y; call them X_B, Y_B.

2. Train a classification tree f_b on X_B, Y_B.

After training, predictions for unseen samples x' can be made by averaging predictions from all the individual regression trees on x':

$$\hat{f} = \frac{1}{B}\sum_{b=1}^{B} f_b(x') \tag{3.3}$$

An estimate of the uncertainty of the prediction can be calculated as the standard deviation of the predictions from all individual regression trees on x':

$$\sigma = \sqrt{\frac{\sum_{b=1}^{B}(f_b(x') - \hat{f})^2}{B-1}} \tag{3.4}$$

Provided with the dataset, Random Forest creates decision trees for randomly selected data samples, obtains decisions from each tree and chooses the best solution utilizing voting. The more trees the Random Forest has, the more robust solution it provides. Since the majority of machine learning tasks are based on classification and regression, Random Forest has been of frequent use by ML specialists. The biggest advantage of the Random Forest classifier is that its default hyperparameters tend to produce pretty good results and also RF does not overfit the model it is building on [14, 17]. RF also uses Gini index [19] to decide how nodes on a decision tree branch based on their probability of occurring [72].

However, there is one particular disadvantage of an RF classifier: a rather complex tree can make the algorithm too slow and inefficient for real-time predictions. Even though Random Forest is fast to train, it takes a lot of time to create predictions once they are trained. For this reason, it is recommended to look for other approaches when run-time performance is essential [17].

Logistic Regression is the baseline supervised machine learning algorithm for classification tasks in Natural Language Processing (NLP). It is an extension of a linear regression model. In a linear model, the relationship between outcome and features can be described by using the following linear equation [21]:

$$\hat{y}^{(i)} = \beta_0 + \beta_1 x_1^{(i)} + ... + \beta_p x_p^{(i)} \tag{3.5}$$

Since the probability value for classification is expected to be between the values 0 and 1, the right side of the equation is wrapped into logistic function:

$$logistic(\eta) = \frac{1}{1 + exp(-\eta)} \tag{3.6}$$

Finally, the probability formula looks like:

$$P(y^{(i)} = 1) = \frac{1}{1 + exp(-(\beta_0 + \beta_1 x_1^{(i)} + ... + \beta_p x_p^{(i)}))} \tag{3.7}$$

Logistic Regression is known to have a close relationship to neural networks. Linear regression is limited by its ability to interpolate between the points, which cannot be interpreted as a probability. For this reason, logistic regression was suggested as a solution for the classification task. Rather than fitting a straight line or hyperplane between given points, Logistic Regression makes use of logistic function [21].

One of the merits of logistic regression is that it can not only classify the data but provide probabilities for sample data too. On top of that, logistic regression can be easily extended from binary classification algorithm to multi-class classification [27]. But, on the other hand, LR is known to have several demerits. One of them is the restrictive nature of expressiveness when for example, interactions must be added manually. Another downside of an algorithm is a multiplicative interpretation of the weights, rather than additive, which interprets the model more difficult [21].

Support Vector Machine is another supervised machine learning algorithm, mainly used for classification and regression tasks. The motivation behind SVM lies in successfully finding (p-1)-dimensional hyperplane, separating data points represented in p-dimensional vector form (a list of p numbers). For instance, provided with data points for the training process, SVM tries to find a decision boundary in form of a line for 2-dimensional data, and a plane for 3-dimensional data, etc. to categorize n-dimensional space into classes [76, 28]. Among many hyperplanes that might classify the data, a reasonable choice as the best hyperplane comes to the one representing the largest separation, or margin between two classes. Advantages of Support Vector Machine [69, 28, 16] are:

- small input sample required for training
- high speed and memory efficiency
- effectiveness in high dimensional space

- versatility: different Kernel functions can be specified for the decision function

Suppose a training dataset of n points is given in the form

$$(x_1, y_1), ..., (x_n, y_n)$$

with y_i being either 1 or -1, indicating which class x_i belongs to. Each x_i is p-dimensional real vector. The goal is to find "maximum-margin hyperplane" separating group of points x_i for which $y_i = 1$ from a group of points x_i for which $y_i = -1$. "Maximum-margin hyperplane" should be such with distance between hyperplane and the nearest point x_i from either group is maximized.

Any hyperplane can be represented as the set of points x satisfying

$$\mathbf{w}^T x - b = 0$$

where $\mathbf{w}$ is the normal vector to the hyperplane. The parameter $\frac{b}{\|\mathbf{w}\|}$ determines the offset of the hyperplane from the origin along the normal vector $\mathbf{w}$.

In case the training data is linearly separable, two parallel hyperplanes that separate two classes of data are selected, so that the distance between them is as large as possible. The area enclosed between two hyperplanes is called "margin" and the maximum margin hyperplane is the one that lies halfway between them. If the dataset is normalized, these hyperplanes can be described by the equation:

$$\mathbf{w}^T x - b = 1$$

any point on or above this boundary belongs to one class, with label 1 and

$$\mathbf{w}^T x - b = -1$$

any point on or below this boundary belongs to another class, with label -1.

Applying the rules of geometry, the distance between two boundaries is $\frac{2}{\|\mathbf{w}\|}$, in order to maximize the distance between hyperplanes, it is required to minimize $\|\mathbf{w}\|$. Distance is calculated using distance from a point to a plane equation. It is essential to ensure the data points do not fall into the margin by adding the next constraint: for each i either

$$\mathbf{w}^T x - b \geq 1, \text{if } y_i = 1$$

or

$$\mathbf{w}^T x - b \leq -1, \text{if } y_i = -1$$

Following constraints state that each data point must lie on the correct side of the margin. This can be rewritten as:

$$y_i(\mathbf{w}^T x_i - b) \geq 1 \text{ , for all } 1 \leq i \leq n$$

Putting this together to obtain optimization problem:

"Minimize $\|\mathbf{w}\|$ subject to $y_i(\mathbf{w}^T \mathbf{x}_i - b) \geq 1$ for $i = 1, ..., n$."

The $\mathbf{w}$ and b that solve this problem determines classifier, $\mathbf{x} \mapsto sgn(\mathbf{w}^T \mathbf{x} - b)$ where $sgn(\cdot)$ is the sign function.

An important consequence of this geometric description is that "maximum-margin hyperplane" is completely determined by those $\vec{x_i}$ that lie nearest to it. These $\mathbf{x}_i$ are called support vectors [28].

Naive Bayes is a probabilistic machine learning algorithm based on the Bayes Theorem. It is one of the most popular classifiers. The fundamental NB assumption is that each feature makes an independent and equal contribution to the outcome [24]. Bayes Theorem calculates the probability of an event occurring given the probability of another event that has already occurred. In a classification task, it assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{3.8}$$

Given a problem instance to be classified, represented by a vector $x = (x_1, ..., x_n)$ representing some n features(independent variables), it assigns to this instance probabilities

$$p(C_k|x_1, ..., x_n) \tag{3.9}$$

for each of K possible outcomes or classes C_k.

Using Bayes' theorem, the conditional probability can be decomposed as:

$$p(C_k|x) = \frac{p(C_k)p(x|C_k)}{p(x)} \tag{3.10}$$

Correspondence of following theorem to plain English can be stated as:

$$posterior = \frac{prior * likelihood}{evidence}$$

Since denominator value does not depend on C and the values of the features x_i are provided, denominator is a constant value, whereas the numerator of the fraction serves as the main interest. The numerator is equivalent to joint probability model:

$$p(C_k, x_1, ..., x_n)$$

which can be rewritten as follows, using the chain rules for repeated applications of the definition of the conditional probability:

$$\begin{aligned} p(C_k, x_1, ..., x_n) &= p(x_1, ..., x_n, C_k) \\ &= p(x_1|x_2, ..., x_n, C_k)p(x_2, ..., x_n, C_k) \\ &= p(x_1|x_2, ..., x_n, C_k)p(x_2|x_3, ..., x_n, C_k)p(x_3, ..., x_n, C_k) \\ &= ... \\ &= p(x_1|x_2, ..., x_n, C_k)p(x_2|x_3, ..., x_n, C_k)...p(x_{n-1}|x_n, C_k)p(x_n|C_k)p(C_k) \end{aligned} \tag{3.11}$$

Now, let us apply "naive" conditional independence assumptions to it, by assuming all features in x to be mutually independent, conditional on the category C_k. Under this assumption:

$$p(x_i|x_{i+1}, ..., x_n, C_k) = p(x_i|C_k) \tag{3.12}$$

In this way, the joint model can be expressed as:

$$\begin{aligned} p(C_k|x_1, ..., x_n) &\propto p(C_k, x_1, ..., x_n) \\ &\propto p(C_k)p(x_1|C_k)p(x_2|C_k)p(x_3|C_k)... \\ &\propto p(C_k)\prod_{i=1}^{n} p(x_i|C_k) \end{aligned} \tag{3.13}$$

where $\propto$ means proportionality.

Taking into account independence assumption mentioned above, conditional distribution over the class variable C is:

$$p(C_k|x_1, ..., x_n) = \frac{1}{Z}p(C_k)\prod_{i=1}^{n} p(x_i|C_k) \tag{3.14}$$

where the evidence $Z = p(x) = \sum_k p(C_k)p(x|C_k)$ is a scaling factor dependent only on $x_1, ..., x_n$, that is, a constant if the values of feature variables are known [23].

Due to its fast training time and ability to work well on a large datasets, Naive Bayes is used for real-time class prediction in sentiment analysis, spam detection and etc [24, 49, 71].

3.6 TPOT for Optimization

TPOT is considered as a Data Science Assistant that helps scientists find the best possible pipeline for a classification task [51, 67]. It was one of the very first AutoML methods and open-source software packages developed for the data science community [11].

TPOT is based on the genetic programming principle to generate the optimized search space. Genetic programming, in its turn, reflects the process of natural selection where the fittest individuals are selected for reproduction in order to produce offspring of the next generation [12].

- Selection phase chooses the fittest individuals and lets them pass their genes to the next generation
- Crossover selects the fittest individuals from above and performs crossover between them to generate a new population
- Mutation of individuals generated y crossover for further random modifications. It is repeated for a few steps or until the best generation is achieved [12, 51]

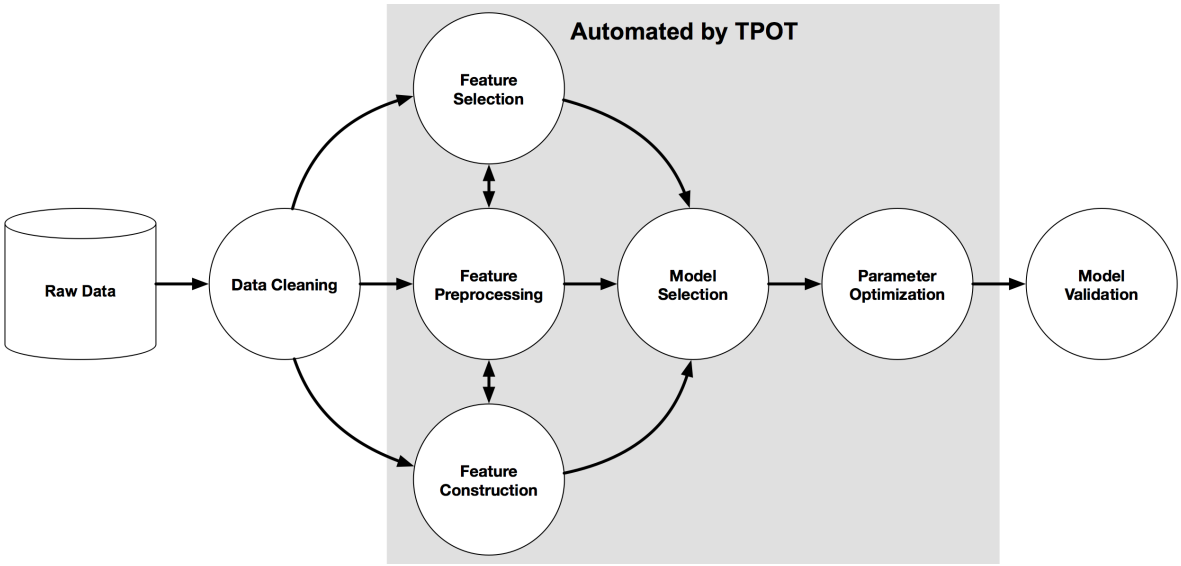

Figure 3.5: An example machine learning pipeline [51]

Figure 3.5 shows an example of ML pipeline and how does TPOT automates feature selection, preprocessing and construction part of it, choosing the best model

afterwards and performing parameter optimization. During the training process, TPOT tries one pipeline, assesses its performance and makes random changes to the pipeline's parameters in search of a better solution. By this, the tool saves up time for scientists, who would have to perform tedious feature engineering for a long span of time. Once the training process is complete, TPOT provides a Python code for the best pipeline, which can be exported by the user [29].

For example, for a 10,000-pipeline configuration, TPOT will evaluate them using 10-fold cross-validation, and therefore resulting in 100,000 models being fit and evaluated on the training data in one grid search. By offering a specific pipeline as a solution for a given problem, TPOT can enable researchers to take a new look at pipeline configurations, which they might not have considered if they did not use it.

TPOT is implemented with a number of configurations, working best for specific tasks. For this research, I made use of Default TPOT configuration, however, other configurations are worth looking at:

- TPOT light uses simple operators in pipelines. In addition to it, this configuration verifies operators to be fast-executing
- TPOT MDR is suitable for problems in the bioinformatics area, with configuration being ideal for genome-wide association studies
- TPOT spare configuration works best for sparse matrices
- TPOT NN is helpful for exploitation of neural network estimators with default POT. Estimators are written in PyTorch
- TPOT cuML is applicable for medium or large-size datasets to search for best pipelines over a limited configuration utilizing the GPU-accelerated estimators [18]

For this research, I employed 13 features chosen by Tranalyzer2 and augmented with the flow entropy, splitting training and testing datasets into 12000 and 9000 flows respectively (similar to experiments carried out on Weka). The ratio of training vs. testing datasets can be set to 3:1 or 1:1, depending on the user's preference. To keep consistency in my research work and be able to compare the AutoML tool

#	Classifier	Preprocessor
1	Bernoulli Naive Bayes	Binarizer
2	Gaussian Naive Bayes	Fast ICA
2	Multinomial Naive Bayes	Feature Agglomeration
3	Decision Tree	Max Abs Scaler
4	Extra Trees	Min Max Scaler
4	Random Forest	Normalizer
5	Gradient Boosting	Nystroem
6	K-Neighbors	PCA
7	Linear SVC	Polynomial Features
8	Logistic Regression	RBF Sampler
9	XGB	Robust Scaler
10	SDG	Standard Scaler
11	MLP (Neural Network)	Zero Count
12		One Hot Encoder

Table 3.8: The current list of classifiers and preprocessors implemented in TPOT

with a standard ML classifier, I provided the same testing datasets for performance assessment.

The number of pipelines TPOT considers depends on the parameters set by the user. The formula used by the TPOT tool for the pipeline number identification is following:

$$population\_size + generations * offspring\_size \tag{3.15}$$

where *population_size* is the number of individuals to keep in GP population after every generation, *generations* is the number of iterations to run pipeline optimization process and $offspring\_size$ is the number of offspring to generate in each GP generation. By default, offspring size is equal to population size and population size is set to 100, unless specified by the user. Setting *verbosity* parameter to 2 shows a progress bar during the training process. In this thesis, I evaluated TPOT's performance over 10, 50, 100 and 200 generations, processing 1100, 5100, 10100 and 20100 pipelines accordingly. Since TPOT's algorithm optimization follows stochastic nature, the AutoML tool will not generate the same pipeline as the best classifier twice. However, if *random_state* parameter is set before the training process starts, it enables the tool to choose the same algorithm when running the training model multiple times.

The list of classifiers and preprocessors implemented in TPOT is quite impressive. Table 3.8 shows current options implemented in the latest version of TPOT tool [29].

However, if the user does not want the tool to consider all implemented classifiers during training, he/she can limit the algorithms and parameters TPOT looks at. The tool provides flexibility to the user to set up custom configurations and parameters. Using this possibility, I configured TPOT to consider only Neural Network classifiers (namely Multi-Layer Perceptron) in the second part of TPOT related experiments to compare pipeline configurations in the end.

3.7 Summary

Setting up the right path to carry the research was a top priority at the start of this research. Once the papers were analyzed, finding publicly available datasets was the next step. While looking for suitable datasets, various factors were taken into account. In particular, the availability of a dataset, the year of release, and the amount of research work being carried using a dataset were all taken into consideration. While some datasets were captured for a fixed period, the capture of others (ImpactGT) is continuing. Before choosing network flow exporters as the next milestone in research, comparing their performance based on scientific work was essential. The choice was narrowed to Tranalyzer2, Argus and DoHlyzer, but taking into account the limitation of the DoHlyzer tool, it was decided to keep the first two network flow extractor tools.

Data pre-processing and feature engineering was the following stage in research analysis. Running benign and malicious PCAP files through Tranalyzer2, and then Argus, labelling and preparing training and testing datasets for the classification task, comparing the learner's performance without entropy were the first several steps in my proposed approach. Right after that, writing a Matlab script for entropy calculation, involving the T-Shark tool for field extraction and analyzing how Decision Tree acted differently provided me more insight regarding the path I needed to follow since the start of the research.

Following the success of the C4.5 classifier, running datasets on other ML classifiers took place. Post-training evaluation and further research were carried out. Finally, the TPOT-AutoML system was used to optimize the proposed approach.

Chapter 4

Evaluations and Results

As discussed earlier, the goal of the thesis is to explore the use of entropy of a network flow to augment statistical flow features to identify malicious DoH tunnels. Proposing highly effective (in terms of F1-measure) performance with a relatively reasonable computational cost solution is a key focus throughout the research methodology. All the experiments were run on a MacBook Pro with a 2.3GHz 8-core Intel Core i9. Thus, the performance of the proposed approach is measured by the following metrics:

1. *Precision* is the ratio of correctly predicted malicious (benign) flows to the total number of malicious (benign) flows.

$$P = \frac{TP}{TP + FP}$$

2. *Recall* is the ratio of correctly predicted malicious (benign) flows to all flows in actual class.

$$R = \frac{TP}{TP + FN}$$

3. *F1-measure* of the weighted average of Precision and Recall.

$$F = \frac{2RP}{R + P}$$

4.1 C4.5 classifier results obtained by using Weka

Once the training dataset is formed and the C4.5 decision tree classifier is trained using different entropy scenarios, five new datasets are used for testing the trained models. It should be noted here that WEKA [1], the open-source ML software is used for training and testing all classifiers using default parameters unless depicted differently. Table 4.1 shows the results of the C4.5 decision tree classifier on training and testing datasets, using flow features augmented by the entropy calculated over all

[1]WEKA - https://www.cs.waikato.ac.nz/ml/weka/

packets of a flow. As it can be seen from the results, the performance of this trained model is still over 90% for the first four test datasets that were not seen during the training. The only exception is the last test dataset containing 9000 malicious flows from DoHBrw.

Scenario	Datasets	P	R	F
Training	DoHBrw+ImpactGT+CICIDS	0.997	0.997	0.997
Testing	DoHBrw benign and attack	0.921	0.906	0.906
	ImpactGT attack	1.000	0.999	1.000
	DoHBrw benign	1.000	0.991	0.995
	CICIDS benign	1.000	0.990	0.995
	DoHBrw attack	1.000	0.821	0.902

Table 4.1: C4.5 classification results of training and test datasets - entropy calculated over all packets of a flow

On the other hand, when no entropy is used to augment flow features, the performance of the classifier drops for all datasets, except the last one, Table 4.3. Specifically, Recall values decline on almost all test datasets, in particular to around 71% and 91% for CICIDS benign and DoHBrw attack datasets, respectively. These results indicate the effectiveness of the entropy for augmenting flow statistical features. Moreover, Table 4.2 shows the results of the C4.5 decision tree classifier on training and testing datasets, using flow features augmented by entropy calculated over the first four packets of a flow. These results demonstrate that only the first four (or less) packets of a flow seem to be enough to calculate the entropy. This not only augments flow statistical features well without decreasing the F-measure, Precision and Recall metrics but also it is computationally less expensive relative to calculating entropy over all packets of a flow when the flow includes more than four packets. It is noticeable that this approach improves the performance of the C4.5 model: one can see an increase of 1% for test datasets containing 9000 flows of benign and malicious flows from DoHBrw and a 1% increase for CICIDS benign datasets. Thus, I propose this model in the following evaluations.

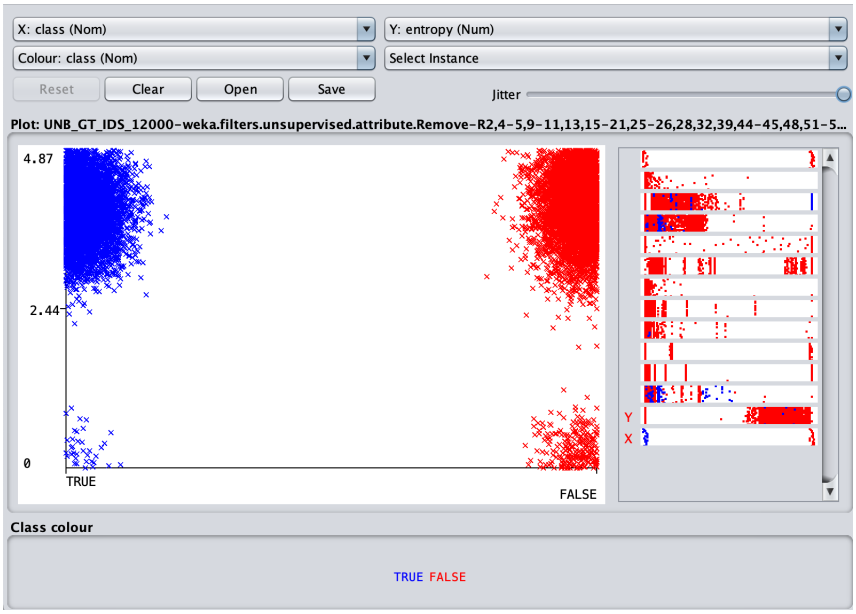

Figure 4.1: Visualization of entropy values distribution over classes in the training dataset using Weka

Scenario	Datasets	P	R	F
Training	DoHBrw+ImpactGT+CICIDS	0.997	0.997	0.997
Testing	DoHBrw benign and attack	0.928	0.917	0.916
	ImpactGT attack	1.000	0.999	1.000
	DoHBrw benign	1.000	0.983	0.983
	CICIDS benign	1.000	1.000	1.000
	DoHBrw attack	1.000	0.828	0.906

Table 4.2: C4.5 classification results of training and test datasets - entropy calculated over the first 4 packets of a flow

Scenario	Datasets	P	R	F
Training	DoHBrw+ImpactGT+ CICIDS	0.956	0.952	0.952
Testing	DoHBrw benign and attack	0.919	0.916	0.916
	ImpactGT attack	1.000	0.985	0.992
	DoHBrw benign	1.000	0.928	0.963
	CICIDS benign	1.000	0.708	0.829
	DoHBrw attack	1.000	0.907	0.951

Table 4.3: C4.5 classification results of training and test datasets - no entropy deployed

4.2 Results of running four ML classifiers on Weka

Table 4.4 presents a comparison of the proposed model using the C4.5 decision tree classifier against Random Forest, Logistic Regression, Support Vector Machine and Naive Bayes classifiers. These ML classifiers are chosen for further evaluations since they were used in literature, as discussed in Chapter 2.

ML Classifiers	Random Forest			SVM			Logistic Regression			Naive Bayes		
	P	R	F	P	R	F	P	R	F	P	R	F
Training: DoHBrw+ImpactGT+CICIDS	1.000	1.000	1.000	0.893	0.870	0.868	0.963	0.962	0.962	0.861	0.818	0.813
Test: DoHBrw benign and attack	0.926	0.917	0.917	0.813	0.750	0.737	0.951	0.951	0.951	0.786	0.730	0.717
Test: ImpactGT attack	1.000	1.000	1.000	1.000	0.983	0.992	1.000	0.999	0.999	1.000	0.984	0.992
Test: DoHBrw benign	1.000	0.989	0.995	1.000	0.785	0.880	1.000	0.890	0.942	1.000	0.543	0.704
Test: CICIDS benign	1.000	1.000	1.000	1.000	0.865	0.928	1.000	0.844	1.000	1.000	0.672	0.804
Test: DoHBrw attack	1.000	0.844	0.915	1.000	0.985	0.992	1.000	0.953	0.976	1.000	0.977	0.988

Table 4.4: Classification results of RF, SVM, LR, and NB on training and test datasets - entropy calculated over first 4 packets of a flow

In all cases, classifiers are evaluated using the same training and test datasets as well as the same feature set, i.e. flow statistical features augmented with the flow entropy feature calculated over the first four packets of a flow. The results show that the Random Forest classifier demonstrates a similar performance as the C4.5 classifier. It is interesting to see that LR, SVM and NB perform pretty well in detecting malicious flows as well. However, they misclassify the majority of benign flows in test datasets. The NB classifier, for example, managed to classify only around 54% of benign flows in the DoHBrw benign dataset, even though it detected almost 98% of attacks. This seems to support the argument Singh et. al. made in [75] regarding the lagging feature of the NB classifier.

Finally, Table 4.5 shows the performance of the RF classifier as I increase the number of trees during its training (the same feature set and the same training dataset as before) using *numIterations* parameter in WEKA. The results show that as the number of trees increases from 100 to 2000, the performance seems to stay pretty much consistent.

Dataset	Training			Testing														
	DoHBrw+ImpactGT +CICIDS			DoHBrw benign and attack			ImpactGT attack			DoHBrw benign			CICIDS benign			DoHBrw attack		
	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F
100	1.000	1.000	1.000	0.926	0.917	0.917	1.000	1.000	1.000	1.000	0.989	0.995	1.000	1.000	1.000	1.000	0.844	0.915
500	1.000	1.000	1.000	0.940	0.935	0.935	1.000	1.000	1.000	1.000	0.990	0.995	1.000	1.000	1.000	1.000	0.880	0.936
1000	1.000	1.000	1.000	0.941	0.935	0.935	1.000	1.000	1.000	1.000	0.990	0.995	1.000	1.000	1.000	1.000	0.881	0.937
1500	1.000	1.000	1.000	0.938	0.932	0.932	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.878	0.935
2000	1.000	1.000	1.000	0.938	0.932	0.932	1.000	1.000	1.000	1.000	0.990	0.995	1.000	1.000	1.000	1.000	0.877	0.935

Table 4.5: Classification results of RF as number of trees increases - entropy calculated over the first 4 packets of a flow

Table 4.6 presents the depth of RF trees, again as I increase the number of trees during training. This indicates that RF trees are deep and therefore resulting in a rather complex RF classifier, with the depth of the tree starting from 24 and going up as the tree size increases. On the other hand, the C4.5 decision tree classifier's trained model is less complex (depth=10) and can be visualized in Figure 4.3. This also demonstrates that Decision Tree trained model uses the entropy attribute throughout the tree to classify a flow as benign or malicious. This supports my hypothesis of using the entropy of a network flow to augment statistical features for identifying malicious behaviours in encrypted tunnels.

Number of Trees	Depth per Tree
100 500	24
1000 1500 2000	26

Table 4.6: Depth of RF per tree as the number of trees increase - Trained on DoHBrw+ImpactGT+CICIDS

Based on these results obtained, using the proposed Decision Tree solution as a predictive model will enable to label the new/unseen flows by providing 13 statistical features augmented with the entropy feature (over the first four packets of a flow). Figure 4.2 illustrates how the final trained model can be used as a predictive tool for the DoH traffic classification.

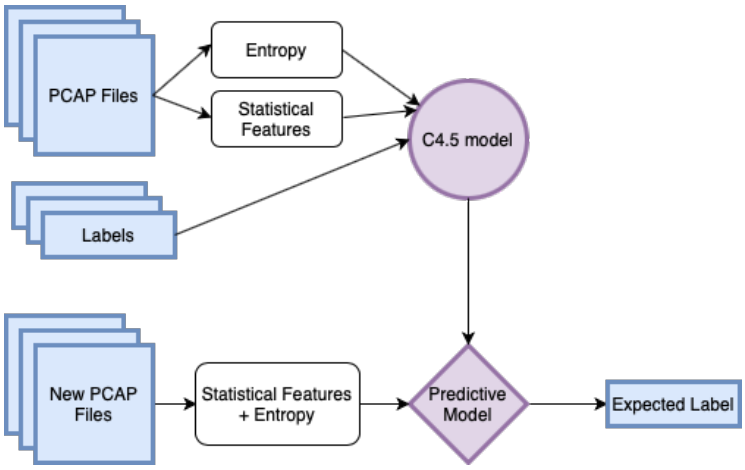

Figure 4.2: Deploying the trained Supervised Learning model as a Predictive Model

4.3 Results of experiments with Argus flow extractor

After proposing a model where entropy was calculated over the first four packets of a flow and running various experiments on Weka, it was decided to take a look at how the Argus tool would perform for the DoH tunnelling identification task. The methodology for working with Argus flow extractor was completely similar to the Tranalyzer2. Table 4.7 compares the results of two tools and how Argus was lagging in terms of performance metrics. Particularly, in the last testing dataset - DoHBrw attack, Argus presented 58.8% Recall value compared to 82.8% for Tranalyzer2. The second tool, on the other hand, overtook the first one for DoHBrw benign and attack dataset, demonstrating a 100% precision rate against 93%. However, considering the two sides of the argument indicates towards giving preference for Tranalyzer2, since the 25% difference in the DoHBrw attack dataset seemed more persuading than 7% in DoHBrw benign and attack dataset.

Dataset	Tranalyzer2			Argus		
	P	R	F	P	R	F
Training: DoHBrw + ImpactGT + CICIDS	0.997	0.997	0.997	0.999	0.999	0.999
Test: DoHBrw benign and attack	0.928	0.917	0.916	0.996	0.965	0.965
Test: ImpactGT attack	1.000	1.000	1.000	1.000	0.997	0.999
Test: DoHBrw benign	1.000	0.983	0.992	1.000	1.000	1.000
Test: CICIDS benign	1.000	1.000	1.000	1.000	0.993	0.996
Test: DoHBrw attack	1.000	0.828	0.906	1.000	0.588	0.740

Table 4.7: Comparison of classification results for Tranalyzer2 and Argus

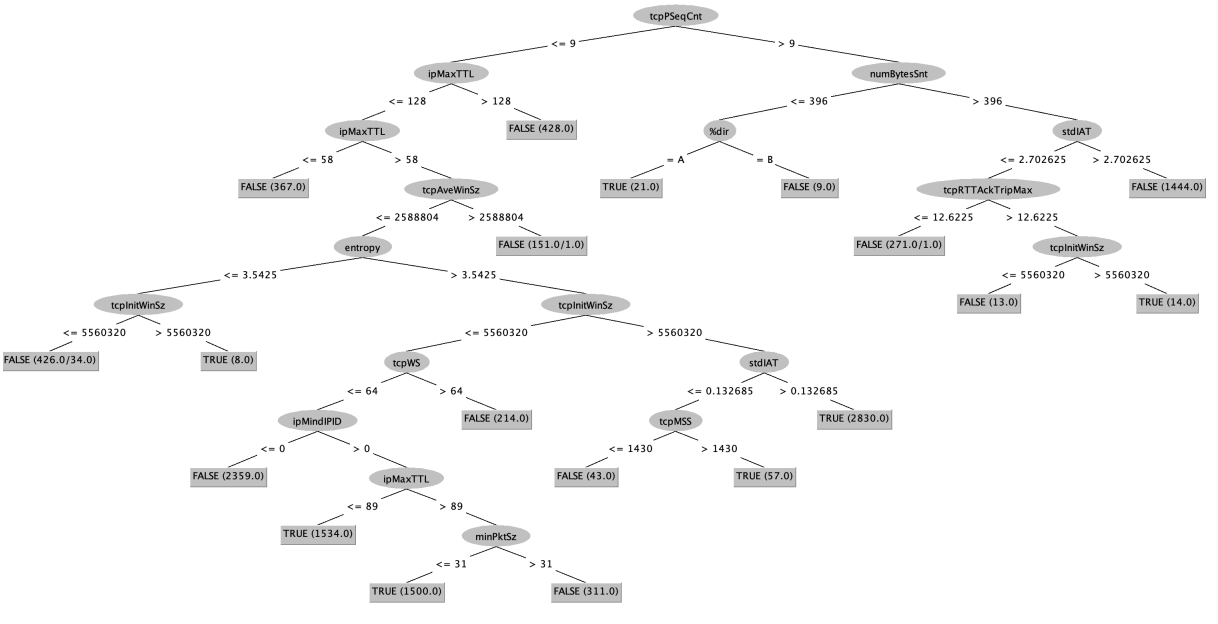

Figure 4.3: Visualization of C4.5 Decision Tree - entropy calculated over the first 4 packets of a flow

4.4 Results of running TPOT-AutoML

Following a discussion in the methodology section regarding the TPOT tool and how it can help find the best classifier for DoH tunnelling detection, I ran 13 features from my predictive model. It needs to be mentioned that the more pipelines TPOT try out, the more time it requires for the training process. TPOT's performance is represented in the same metrics used in the first part of the research.

Firstly, I configured TPOT to use all 11 classifiers in search of the best solution, without limiting its choice. Table 4.8 presents the results of running one training and five testing datasets. It is interesting to see that for the smallest number of generations (n=10), TPOT chose RF classifier, achieving almost 100% training CV score. As the number of generations goes up, the choice of classifier shifts to GB classifier, reaching the same mark. Following this work, I configured TPOT to use a Neural Network classifier only, looking at how performance will change over the course of generation number increase (see Table 4.9). In this case, MLP classifier demonstrated 88% CV score for n=10 and 90% CV score for n=200. However, the performance of the trained model on testing datasets was quite low. Parameters set

during the training process are described in detail in the Methodology chapter.

# of generations	10			50			100			200		
Classifier chosen	Random Forest			Gradient Boosting			Gradient Boosting			Gradient Boosting		
Training CV score(avg)	0.997			0.998			0.998			0.998		
Testing datasets	P	R	F	P	R	F	P	R	F	P	R	F
DoHBrw benign and attack	1.00	0.92	0.96	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99
ImpactGT attack	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
CICIDS benign	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
DoHBrw benign	1.00	0.98	0.99	1.00	0.99	0.99	1.00	0.98	0.99	1.00	0.99	0.99
DoHBrw attack	1.00	0.92	0.96	1.00	0.98	0.99	1.00	0.99	1.00	1.00	0.99	1.00

Table 4.8: Results of running TPOT (11 classifiers) for optimizing the proposed approach

# of generations	10			50			100			200		
Classifier chosen	MLP			MLP			MLP			MLP		
Training CV score(avg)	0.877			0.873			0.878			0.893		
Testing datasets	P	R	F	P	R	F	P	R	F	P	R	F
DoHBrw benign and attack	0.69	0.68	0.67	0.44	0.49	0.36	0.69	0.66	0.65	0.73	0.71	0.71
ImpactGT attack	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
CICIDS benign	1.00	0.85	0.92	1.00	0.72	0.84	1.00	0.85	0.92	1.00	0.86	0.92
DoHBrw benign	1.00	0.58	0.73	1.00	0.94	0.97	1.00	0.43	0.60	1.00	0.56	0.72
DoHBrw attack	1.00	0.84	0.91	1.00	0.04	0.08	1.00	0.87	0.93	1.00	0.88	0.94

Table 4.9: Results of running TPOT - Neural Network classifiers only

A rather interesting idea came once the results from applying the TPOT tool were received. I compared the performance of the Random Forest classifier both on Weka and TPOT tools. As it was mentioned in the methodology, the RF classifier was set with default parameters in Weka, while TPOT came up with a tailored pipeline for the RF algorithm. While training results were almost identical, testing datasets illustrated some differences (Table 4.10). Precision value of DoHBrw benign and attack dataset in Weka was lagging behind TPOT's performance by almost 7%, as for DoHBrw attack dataset, TPOT outperformed Weka by 7% in Recall and 4% in F-1 score. The following results prove TPOT's developers' core principles: trying out different pipeline configurations can substantially improve the classifier's performance since both tools were provided with the same datasets, consisting of the same labelled flows.

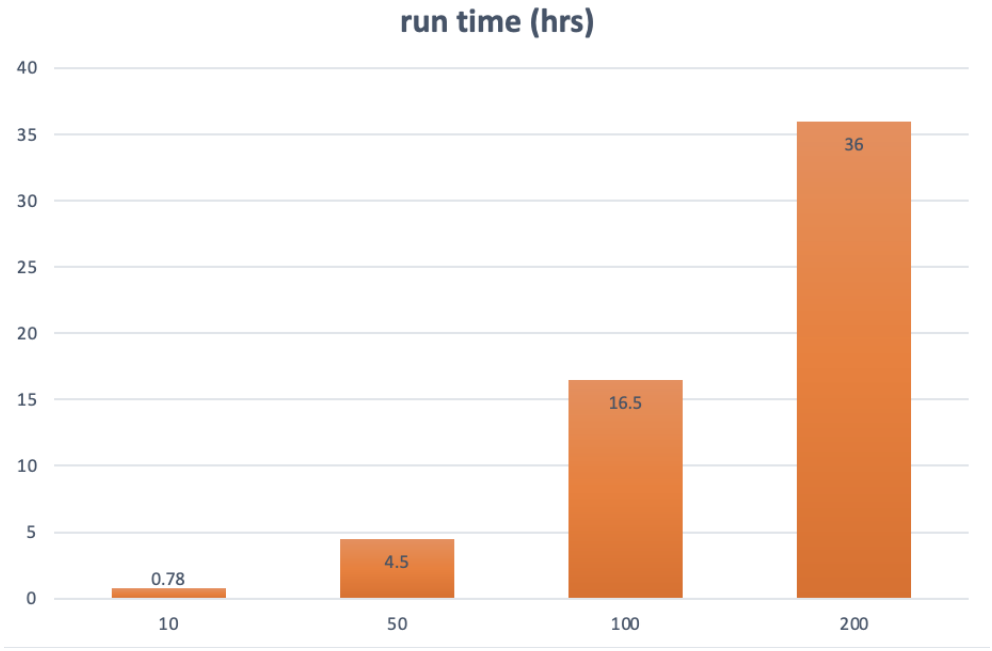

Figure 4.4: Computational cost (hrs) of TPOT during training as the number of generations increase where TPOT considered all 11 classifiers

Tools	Weka			TPOT		
	P	R	F	P	R	F
Training	1.00	1.00	1.00		0.997	
Test:DoHBrw benign and attack	0.926	0.917	0.917	1.00	0.92	0.96
Test:ImpactGT attack	1.00	1.00	1.00	1.00	1.00	1.00
Test:CICIDS benign	1.00	1.00	1.00	1.00	1.00	1.00
Test:DoHBrw benign	1.00	0.989	0.995	1.00	0.98	0.99
Test:DoHBrw attack	1.00	0.844	0.915	1.00	0.92	0.96

Table 4.10: Comparison of Weka RF classifier versus TPOT

4.5 Summary

As the results of experiments demonstrate, the target set at the start of the research was achieved with justifications provided. Indeed, when it comes to feature extraction from network traffic, Tranalyzer2 performed better compared to Argus and DoHlyzer, deploying only seven features out of 109 for anomaly detection with no entropy calculated (Table 3.2). After that, the idea of utilizing network flow entropy was advocated by the results obtained (Table 4.1 vs. Table 4.3). Once the network flow

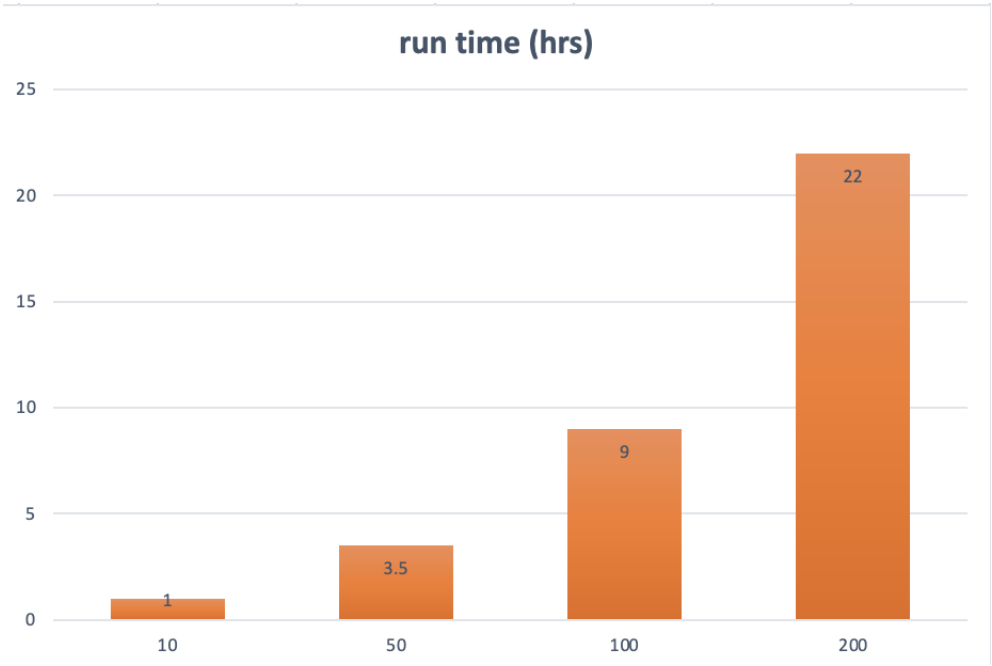

Figure 4.5: Computational cost (hrs) of TPOT during training as the number of generations increase where TPOT considered Neural Network classifiers only

entropy approach was adopted, the goal of improving the model's complexity and the computational cost was established. Augmenting entropy over the first four packets with other statistical features verified my assumptions since the model's performance did not decrease on testing datasets (Table 4.5). Comparing C4.5's performance with other four ML classifiers was outlined as the baseline: RF, LR, SVM and NB algorithms demonstrated their ability to separate malicious DoH/DNS flows from benign ones (Table 4.4). C4.5 Decision Tree provided a less complex model, compared to a better performing RF algorithm (tree depth of 10 against 24). To explore the most suitable choice of flow extractor, similar experiments were carried out using the Argus tool. As a result, Tranalyzer2 outperformed Argus in the DoHBrw attack dataset by a large margin (Table 4.7). Finally, training the TPOT-AutoML system with the model of the proposed approach and testing it on five datasets not only provided the optimized model for the proposed approach but also demonstrated the effectiveness and the efficiency of the optimized model in DNS malicious tunnel behaviour detection.

Chapter 5

Conclusion

With the mounting criticism against DoH protocol, which can allow attackers to bypass organizational controls, new approaches to monitoring encrypted DNS queries, such as DoH, are necessary. Researching this area has been a challenging task due to several limitations. First of all, finding publicly available datasets built on new DoH protocol with malicious activities was the first obstacle I encountered. Limited tools used for feature extraction from network flows were the second difficulty standing in the way. Some recent work proposed the DoHlyzer tool to analyze DoH traffic whereas other works employed network packet entropy to address the aforementioned challenges.

In this thesis, exploring a solution for these challenges without performing deep packet inspection, payload or metadata analysis was an important goal set at the beginning. To this end, I studied the use of the concept of "entropy of a flow" to augment flow statistical features for identifying malicious DoH tunnels. To achieve this, a thorough investigation of the use of different flow exporters was performed. The flow exporters that were analyzed include Argus, DoHlyzer and Tranalyzer2. Results showed the limitation of the DoHlyzer tool in terms of protocols it could support, TCP, but not UDP, and hence making feature extraction for UDP flows impossible. Taking into account these limitations present in the DoHlyzer tool for flow feature extraction, it was decided to consider Tranalyzer2 and Argus tools more closely to select the most suitable statistical flow features for the task. These features were then augmented with flow entropy. To this end, three different ways of calculating the entropy of a flow (over all packets of a flow, over the first 96 bytes of a flow and over the first 4/5/6 packets of a flow) using ML classifiers (DT, RF, LR, SVM and NB) over different datasets were evaluated. The evaluations showed that the C4.5 Decision Tree classifier achieved very high performance in the best case (F-measure 99.7%) when flow statistical features obtained by Tranalyzer2 were augmented with the entropy

feature of a flow calculated over the first 4 packets. This proposed model not only achieved high performance on all datasets employed but also outperformed the model that did not use the entropy feature. Furthermore, heterogeneous aspects of the datasets employed, from different protocols to different packet sizes to different flow characteristics and different behaviours indicated the generalizability of the proposed model over different real-world scenarios.

Moreover, the thesis research demonstrated the effectiveness of the TPOT-AutoML system for optimizing the proposed model to detect malicious DoH flows. Employing TPOT with flow statistical features augmented with entropy calculated over the first four packets enabled me to look at different pipeline configurations at 10, 50, 100 and 200 generations. It should be noted here that with an increase in the number of generations, the training time of TPOT also increases. However, this cost seems to be reasonable to obtain the optimization of the proposed solution. Thus, it is concluded that for a lower number of generations allocated for training, Random Forest outperforms other classifiers implemented in TPOT. If the user is interested in the best performance, then setting a generation number to 100 or 200 is recommended.

Future research will explore the proposed system's behaviour against evasive and adversarial attacks to improve its robustness [62]. Moreover, the proposed model will be evaluated as a predictor on other datasets to investigate its generalization under concept shifts and drifts [55]. Last but not the least, further research into the analysis of DoH and DoT protocols is necessary against the rising privacy and security concerns of our digital world.

Appendix

Flow Features

The tables below provide a list of features extracted by three flow extractor tools deployed in my research. The list of fields extractable by Argus tool is following [1]:

#	Field name	Description
1	srcid	argus source identifier
2	rank	ordinal value of this output flow record i.e. sequence number
3	stime	record start time
4	ltime	record last time
5	trans	aggregation record count
6	flgs	flow state flags seen in transaction
7	seq	argus sequence number
8	dur	record total duration
9	runtime	total active flow run time. This value is generated through aggregation, and is the sum of the records duration
10	idle	time since the last packet activity. This value is useful in real-time processing, and is the current time - last time
11	mean	average duration of aggregated records
12	stddev	standard deviation of aggregated duration times
13	sum	total accumulated durations of aggregated records
14	min	minimum duration of aggregated records
15	max	maximum duration of aggregated records
16	smac	source MAC addr
17	dmac	destination MAC addr
18	soui	oui portion of the source MAC addr
19	doui	oui portion of the destination MAC addr
20	saddr	source IP addr
21	daddr	destination IP addr
22	proto	transaction protocol

#	Field name	Description
23	sport	source port number
24	dport	destination port number
25	stos	source TOS byte value
26	dtos	destination TOS byte value
27	sdsb	source diff serve byte value
28	ddsb	destination diff serve byte value
29	sco	source IP address country code
30	dco	destination IP address country code
31	sttl	src → dst TTL value
32	dttl	dst → src TTL value
33	shops	estimate of number of IP hops from src to this point
34	dhops	estimate of number of IP hops from dst to this point
35	sipid	source IP identifier
36	dipid	destination IP identifier
37	smpls	source MPLS identifier
38	dmpls	destination MPLS identifier
39	autoid	Auto generated identifier (mysql)
40	sas	Src origin AS
41	das	Dst origin AS
42	ias	Intermediate origin AS, AS of ICMP generator
43	cause	Argus record cause code. Valid values are Start, Status, Stop, Close, Error
44	nstroke	Number of observed keystrokes
45	snstroke	Number of observed keystrokes from initiator (src) to target (dst)
46	dnstroke	Number of observed keystrokes from target (dst) to initiator (src)
47	pkts	total transaction packet count
48	spkts	src → dst packet count
49	dpkts	dst → src packet count

#	Field name	Description
50	bytes	total transaction bytes
51	sbytes	src → dst transaction bytes
52	dbytes	dst → src transaction bytes
53	appbytes	total application bytes
54	sappbytes	src → dst application bytes
55	dappbytes	dst →src application bytes
56	pcr	producer consumer ratio
57	load	bits per second
58	sload	source bits per second
59	dload	destination bits per second
60	loss	pkts retransmitted or dropped
61	sloss	source pkts retransmitted or dropped
62	dloss	destination pkts retransmitted or dropped
63	ploss	percent pkts retransmitted or dropped
64	psloss	percent source pkts retransmitted or dropped
65	pdloss	percent destination pkts retransmitted or dropped
66	retrans	pkts retransmitted
67	sretrans	source pkts retransmitted
68	dretrans	destination pkts retransmitted
69	pretrans	percent pkts retransmitted
70	psretrans	percent source pkts retransmitted
71	pdretrans	percent destination pkts retransmitted
72	sgap	source bytes missing in the data stream. Available after argus-3.0.4
73	dgap	destination bytes missing in the data stream. Available after argus-3.0.4
74	rate	pkts per second
75	srate	source pkts per second
76	drate	destination pkts per second
77	dir	direction of transaction

#	Field name	Description
78	sintpkt	source interpacket arrival time (mSec)
79	sintdist	source interpacket arrival time distribution
80	sintpktact	source active interpacket arrival time (mSec)
81	sintdistact	source active interpacket arrival time (mSec)
82	sintpktidl	source idle interpacket arrival time (mSec)
83	sintdistidl	source idle interpacket arrival time (mSec)
84	dintpkt	destination interpacket arrival time (mSec)
85	dintdist	destination interpacket arrival time distribution
86	dintpktact	destination active interpacket arrival time (mSec)
87	dintdistact	destination active interpacket arrival time distribution (mSec)
88	dintpktidl	destination idle interpacket arrival time (mSec)
89	dintdistidl	destination idle interpacket arrival time distribution
90	sjit	source jitter (mSec)
91	sjitact	source active jitter (mSec)
92	sjitidle	source idle jitter (mSec)
93	djit	destination jitter (mSec)
94	djitact	destination active jitter (mSec)
95	djitidle	destination idle jitter (mSec)
96	state	transaction state
97	label	Metadata label
98	suser	source user data buffer
99	duser	destination user data buffer
100	swin	source TCP window advertisement
101	dwin	destination TCP window advertisement
102	svlan	source VLAN identifier
103	dvlan	destination VLAN identifier
104	svid	source VLAN identifier
105	dvid	destination VLAN identifier
106	svpri	source VLAN priority

#	Field name	Description
107	dvpri	destination VLAN priority
108	srng	start time for the filter timerange
109	erng	end time for the filter timerange
110	stcpb	source TCP base sequence number
111	dtcpb	destination TCP base sequence number
112	tcprtt	TCP connection setup round-trip time, the sum of 'synack' and 'ackdat'
113	synack	TCP connection setup time, the time between the SYN and the SYN_ACK packets
114	ackdat	TCP connection setup time, the time between the SYN_ACK and the ACK packets
115	tcpopt	the TCP connection options seen at initiation. The tcpopt indicator consists of a fixed length field, that reports presence of any of the TCP options that Argus tracks
116	inode	ICMP intermediate node
117	offset	record byte offset in file or stream
118	smeansz	Mean of the flow packet size transmitted by the src (initiator)
119	dmeansz	Mean of the flow packet size transmitted by the dst (target)
120	spktsz	histogram for the src packet size distribution
121	smaxsz	maximum packet size for traffic transmitted by the src
122	dpktsz	histogram for the dst packet size distribution
123	dmaxsz	maximum packet size for traffic transmitted by the dst
124	sminsz	minimum packet size for traffic transmitted by the src
125	dminsz	minimum packet size for traffic transmitted by the dst

The list of features extracted by DoHlyzer tool [70]:

#	Field name	Description
1	SourceIP	IP address of source machine
2	DestinationIP	IP address of destination machine

#	Field name	Description
3	SourcePort	Port number of source machine
4	DestinationPort	Port number of destination machine
5	Timestamp	Date time of the flow created
6	Duration	Duration of the network flow
7	FlowBytesSent	Amount of bytes sent from machine used to run DoHlyzer
8	FlowSentRate	Rate of bytes sent in the current flow
9	FlowBytesReceived	Amount of bytes received
10	FlowReceivedRate	Rate of the bytes received in the current flow
11	PacketLengthVariance	Variance of packet length for each flow
12	PacketLengthStandardDeviation	Standard deviation of packet length for each flow
13	PacketLengthMean	Mean of packet length for each flow
14	PacketLengthMedian	Median of packet length for each flow
15	PacketLengthMode	Mode of packet length for each flow
16	PacketLengthSkewFromMedian	Skew of packet length for each flow using median
17	PacketLengthSkewFromMode	Skew of packet length for each flow using mode
18	PacketLengthCoefficientofVariation	Coefficient of variance of a packet lengths list
19	PacketTimeVariance	Variance of packet times for each flow
20	PacketTimeStandardDeviation	Standard deviation of packet times for each flow
21	PacketTimeMean	Mean of packet times for each flow
22	PacketTimeMedian	Median of packet times for each flow
23	PacketTimeMode	Mode of packet times for each flow
24	PacketTimeSkewFromMedian	Skew of packet times for each flow using median

#	Field name	Description
25	PacketTimeSkewFromMode	Skew of packet time for each flow using mode
26	PacketTimeCoefficientofVariation	Coefficient of variance of a packet time list
27	ResponseTimeTimeVariance	Variance of the list of time differences between an outgoing packet and the following response packet
28	ResponseTimeTimeStandardDeviation	The standard deviation of the list of time differences between an outgoing packet and the following response packet
29	ResponseTimeTimeMean	The mean of the list of time differences between an outgoing packet and the following response packet
30	ResponseTimeTimeMedian	The median of the list of tie differences between an outgoing packet and the following response packet
31	ResponseTimeTimeSkewFromMedian	Skew of the list of time differences between an outgoing packet and the following response packet using median
32	ResponseTimeTimeSkewFromMode	Skew of the list of time differences between an outgoing packet and the following response packet using mode
33	ResponseTimeTimeCoefficientofVariation	Coefficient of variance of the list of time differences between an outgoing packet and the following response packet
34	DoH	Boolean defining a packet is DoH or non-DoH

Lastly, the features extracted by Tranalyzer2[9]:

#	Field name	Description
1	%dir	Flow direction
2	flowInd	Flow index
3	flowStat	Flow status and warnings
4	timeFirst	Date time of the first packet
5	timeLast	Date time of the last packet
6	duration	Flow duration
7	numHdrDesc	Number of different headers descriptions
8	numHdrs	Number of headers (depth) in header description
9	hdrDesc	Headers description
10	srcMac	source MAC address
11	dstMac	destination MAC address
12	ethType	Ethernet type
13	ethVlanID	VLAN IDs
14	srcIP	source IP
15	srcIPCC	source IP country
16	srcIPOrg	source IP organization
17	srcPort	source port
18	dstIP	destination IP
19	dstIPCC	destination IP country
20	dstIPOrg	destination IP organization
21	dstPort	destination port
22	l4Proto	Layer 4 protocol
23	macStat	MAC statistics
24	macPairs	MAC pairs
25	srcMac_dstMac_numP	source/destination MAC addresses, number of packets
26	srcManuf_dstManuf	source/destination MAC manufacturers
27	dstPortClassN	port based classification of the destination port number

#	Field name	Description
28	dstPortClass	classification of the destination port
29	numPktsSnt	number of packets sent
30	numPktsRcvd	number of packets received
31	numBytesSnt	number of bytes sent
32	numBytesRcvd	number of bytes received
33	minPktSz	minimum packet size
34	maxPktSz	maximum packet size
35	avePktSize	average packet size
36	stdPktSize	standard packet size
37	minIAT	minimum inter-arrival time
38	maxIAT	maximum inter-arrival time
39	aveIAT	average inter-arrival time
40	stdIAT	standard inter-arrival time
41	pktps	sent packets per second
42	bytps	sent bytes per second
43	pktAsm	packet stream asymmetry
44	bytAsm	byte stream asymmetry
45	tcpFStat	multiple values possible for TCP flag stat
46	ipMindIPID	IP Minimum delta IP Identification
47	ipMaxdIPID	IP Maximum delta IP Identification
48	ipMinTTL	IP Minimum Time to Live (TTL)
49	ipMaxTTL	IP Maximum Time to Live (TTL)
50	ipTTLChg	IP TTL Change Count
51	ipTOS	IP Type of Service
52	ipFlags	IP flags
53	ipOptCnt	IP options count
54	ipOptCpCl_Num	the aggregated IP options are coded as a bit field in hexadecimal notation where the bit position denotes the IP options
55	ip6OptCntHH_D	IPv6 aggregated hop by hop dest option counts
56	ip6OptHH_D	IPv6 hop by hop destination options

#	Field name	Description
57	tcpISeqN	TCP initial sequence number
58	tcpPSeqCnt	TCP packet sequence count
59	tcpSeqSntBytes	TCP sent sequence diff bytes
60	tcpSeqFaultCnt	TCP sequence number fault count
61	tcpPAckCnt	TCP packet ACK count
62	tcpFlwLssAckRcvdBytes	TCP flawless ACK received bytes
63	tcpAckFaultCnt	TCP ACK number fault count
64	tcpInitWinSz	TCP initial effective window size
65	tcpAveWinSz	TCP average effective window size
66	tcpMinWinSz	TCP minimum effective window size
67	tcpMaxWinSz	TCP maximum effective window size
68	tcpWinSzDwnCnt	TCP effective window size change down count
69	tcpWinSzUpCnt	TCP effective window size change up count
70	tcpWinSzChgDirCnt	TCP effective window size direction change count
71	tcpWinSzThRt	TCP packet count ratio below window size WINMIN
72	tcpFlags	TCP aggregated protocol flags (FIN, SYN, RST, PSH, ACK, URG, ECE, CWR)
73	tcpAnomaly	TCP aggregated header anomaly flags
74	tcpOptPktCnt	TCP options packet count
75	tcpOptCnt	TCP options count
76	tcpOptions	TCP aggregated options
77	tcpMSS	TCP maximum segment size
78	tcpWS	TCP window scale factor
79	tcpMPTBF	MPTCP type bitfield
80	tcpMPF	MPTCP flags
81	tcpMPAID	MPTCP address ID
82	tcpMPdssF	MPTCP DSS flags
83	tcpTmS	TCP time stamp
84	tcpTmER	TCP time echo reply

#	Field name	Description
85	tcpEcI	TCP estimated counter increment
86	tcpUtm	TCP estimated up time
87	tcpBtm	TCP estimated boot time
88	tcpSSASAATrip	(A) TCP trip time SYN, SYN-ACK,(B) TCP trip time SYN-ACK, ACK
89	tcpRTTAckTripMin	TCP ACK trip minimum
90	tcpRTTAckTripMax	TCP ACK trip maximum
91	tcpRTTAckTripAve	TCP ACK trip average
92	tcpRTTAckTripJitAve	TCP ACK trip jitter average
93	tcpRTTSseqAA	(A) TCP round trip time SYN, SYN-ACK, ACK (B) TCP round trip time ACK-ACK
94	tcpRTTAckJitAve	TCP ACK round trip average jitter
95	tcpStates	TCP states
96	icmpStat	status
97	icmpTCcnt	type code count
98	icmpBFTypH_TypL_Code	aggregated type H(>128),L(<32) and code bit-field
99	icmpTmGtw	time/gateway
100	icmpEchoSuccRatio	echo reply/request success ratio
101	icmpPFindex	parent flow index
102	connSip	number of unique source IPs
103	connDip	number of unique destination IPs
104	connSipDip	number of connections between source and destination IPs
105	connSipDprt	number of connections between source and destination port
106	connF	the f number, experimental: connSipDprt/connSip

Following is Tshark command executed to extract user-specified fields from network flow packets and write them into separate json file. In this example, the flows represented in pcap file are TCP protocol flows:

```
tshark -T json -e ip.src -e ip.dst -e tcp.srcport -e tcp.dstport -e
frame.time_epoch -e tcp.payload -r DoHBrw_benign.pcap
> DoHBrw_benign.json
```

Benign and Attack Scenarios represented in Datasets

Dataset	Benign	Malicious
DoHBrw	Accessing thousands of websites that use HTTPS protocol from Alexa domain	DNS tunnelling tools as dns2tcp, DNSCat2, Iodine are used to generate malicious DoH traffic. Tools send TCP traffic encapsulated in DNS queries, in other words, they create tunnels of encrypted data
ImpactGT	-	Executing suspect Windows executables in a sterile, isolated environment with a limited access to the Internet
CICIDS	Profiling behaviour of human interactions and generating naturalistic benign background traffic, build abstract behaviour of 25 users	The most common attacks based on 2016 McAfee report, like Web based, Brute Force, DoS, DDoS, Infiltration, Heart-bleed, Bot and Scan

Table 5.4: Benign and Malicious Traffic Representation in datasets supplied

Bibliography

[1] Argus. https://openargus.org/using-argus. Accessed: 20-Sep-2021.

[2] CIRA-CIC-DoHBrw-2020. https://www.unb.ca/cic/datasets/dohbrw-2020.html. Accessed: 10-Oct-2021.

[3] DoHlyzer. https://github.com/ahlashkari/DoHlyzer. Accessed: 10-Oct-2021.

[4] DoHMeter. https://github.com/ahlashkari/DOHlyzer/tree/master/DoHMeter. Accessed: 10-Oct-2021.

[5] Impact Cyber Trust. https://www.impactcybertrust.org. Accessed: 6-Mar-2021.

[6] Tranalyzer. https://tranalyzer.com. Accessed: 19-Sep-2021.

[7] Weka Wiki. https://waikato.github.io/weka-wiki/visualization/extensions_for_wekas_main_gui/. Accessed: 1-Nov-2021.

[8] C4.5 Algorithm. https://en.wikipedia.org/wiki/C4.5_algorithm, 2008. Accessed: Nov-2021.

[9] Tranalyzer - development team. https://www.onworks.net/downloadapp/SOFTWARE/documentation.pdf?service=service01, 2008. Accessed: 28-Nov-2021.

[10] Mathematics behind classification and regression trees. https://stats.stackexchange.com/questions/44382/mathematics-behind-classification-and-regression-trees, 2013. Accessed: 23-Nov-2021.

[11] AutoML: TPOT. http://automl.info/tpot/, 2016. Accessed: Nov-2021.

[12] Introduction to Genetic Algorithms. https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3, 2017. Accessed: 21-Nov-2021.

[13] Machine Learning Classifiers. https://towardsdatascience.com/machine-learning-classifiers-a5cc4e1b0623, 2018. Accessed: Nov-2021.

[14] Understanding Random Forests Classifiers in Python. https://www.datacamp.com/community/tutorials/random-forests-classifier-python, 2018. Accessed: Nov-2021.

[15] Weka Tutorial. https://www.tutorialspoint.com/weka/index.html, 2019. Accessed: 1-Nov-2021.

[16] 1.4 - Support Vector Machines. https://scikit-learn.org/stable/modules/svm.html, 2021. Accessed: 19-Nov-2021.

[17] A complete guide to the Random Forest Algorithm. https://builtin.com/data-science/random-forest-algorithm, 2021. Accessed: Nov-2021.

[18] Automate Machine Learning using TPOT - Explore thousands of possible pipelines and find the best. https://www.analyticsvidhya.com/blog/2021/05/automate-machine-learning-using-tpot%E2%80%8A-%E2%80%8Aexplore-thousands-of-possible-pipelines-and-find-the-best/, 2021. Accessed: 21-Nov-2021.

[19] Gini coefficient. https://en.wikipedia.org/wiki/Gini_coefficient, 2021. Accessed: 17-Nov-2021.

[20] HTTPS encryption on the web. https://transparencyreport.google.com/https/overview?hl=en, 2021. Accessed: 13-Nov-2021.

[21] Interpretable Machine Learning. https://christophm.github.io/interpretable-ml-book/logistic.html, 2021. Accessed: Nov-2021.

[22] J48 Classification in a Nutshell. https://medium.com/@nilimakhanna1/j48-classification-c4-5-algorithm-in-a-nutshell-24c50d20658e, 2021. Accessed: Nov-10-2021.

[23] Naive Bayes Classifier: Wikipedia. https://en.wikipedia.org/wiki/Naive_Bayes_classifier, 2021. Accessed: 21-Nov-2021.

[24] Naive Bayes Classifiers. https://www.geeksforgeeks.org/naive-bayes-classifiers/, 2021. Accessed: Nov-2021.

[25] Random Forest. https://en.wikipedia.org/wiki/Random_forest, 2021. Accessed: 18-Nov-2021.

[26] Solving unintended challenges with DoT and DoH. https://www.infoblox.com/wp-content/uploads/infoblox-solution-note-dot-and-doh-present-new-challenges.pdf, 2021. Accessed: 3-Dec-2021.

[27] Speech and Language Processing. https://web.stanford.edu/~jurafsky/slp3/5.pdf, 2021. Accessed: Nov-2021.

[28] Support-vector machine. https://en.wikipedia.org/wiki/Support-vector_machine, 2021. Accessed: 19-Nov-2021.

[29] Epistasis Lab at UPenn. GitHub repository of TPOT tool. `https://github.com/EpistasisLab/tpot`, 2018. Accessed: 1-Nov-2021.

[30] E. Sandi Aung and H. Yamana. Url-based phishing detection using the entropy of non-alphanumeric characters. In *Proceedings of the 21st International Conference on Information Integration and Web-based Applications & Services, iiWAS 2019, Munich, Germany, December 2-4, 2019*, pages 385–392. ACM, 2019.

[31] Y. M. Banadaki. Detecting malicious dns over https traffic in domain name system using machine learning classifiers. In *Journal of Computer Science and Applications*, pages 46–55. Science and Education Publishing, 2020.

[32] M. Behnke, N. Briner, D. Cullen, K. Schwerdtfeger, J. Warren, R. Basnet, and T. Doleck. Feature engineering and machine learning model comparison for malicious activity detection in the dns-over-https protocol. *IEEE Access*, 9:129902–129916, 2021.

[33] P. Berezinski, J. Pawelec, M. Malowidzki, and R. Piotrowski. Entropy-based internet traffic anomaly detection: A case study. In Wojciech Zamojski, Jacek Mazurkiewicz, Jaroslaw Sugier, Tomasz Walkowiak, and Janusz Kacprzyk, editors, *Proceedings of the Ninth International Conference on Dependability and Complex Systems DepCoS-RELCOMEX. June 30 - July 4, 2014, Brunów, Poland*, volume 286 of *Advances in Intelligent Systems and Computing*, pages 47–58. Springer, 2014.

[34] L. Bernaille and R. Teixeira. Early recognition of encrypted applications. In Steve Uhlig, Konstantina Papagiannaki, and Olivier Bonaventure, editors, *Passive and Active Network Measurement, 8th Internatinoal Conference, PAM 2007, Louvain-la-neuve, Belgium, April 5-6, 2007, Proceedings*, volume 4427 of *Lecture Notes in Computer Science*, pages 165–175. Springer, 2007.

[35] T. Böttger, F. Cuadrado, G. Antichi, E. Leão Fernandes, G. Tyson, I. Castro, and S. Uhlig. An empirical study of the cost of dns-over-https. In *Proceedings of the Internet Measurement Conference, IMC 2019, Amsterdam, The Netherlands, October 21-23, 2019*, pages 15–21. ACM, 2019.

[36] J. Brownlee. 4 Types of Classification Tasks in Machine Learning. `https://machinelearningmastery.com/types-of-classification-in-machine-learning/`, 2020. Accessed: Nov-2021.

[37] K. Bumanglag and H. Kettani. On the impact of DNS over HTTPS paradigm on cyber systems. In *3rd International Conference on Information and Computer Technologies, ICICT 2020, San Jose, CA, USA, March 9-12, 2020*, pages 494–499. IEEE, 2020.

[38] S. Burschka and B. Dupasquier. Tranalyzer: Versatile high performance network traffic analyser. In *2016 IEEE Symposium Series on Computational Intelligence, SSCI 2016, Athens, Greece, December 6-9, 2016*, pages 1–8. IEEE, 2016.

[39] S. Burschka and B. Dupasquier. Tranalyzer: Versatile high performance network traffic analyser. In *2016 IEEE Symposium Series on Computational Intelligence, SSCI 2016, Athens, Greece, December 6-9, 2016*, pages 1–8. IEEE, 2016.

[40] A.J. Campbell and N. Zincir-Heywood. Exploring tunneling behaviours in malicious domains with self-organizing maps. In *2020 IEEE Symposium Series on Computational Intelligence, SSCI 2020, Canberra, Australia, December 1-4, 2020*, pages 1419–1426. IEEE, 2020.

[41] A. Das, M. Shen, M. Shashanka, and J. Wang. Detection of exfiltration and tunneling over DNS. In Xuewen Chen, Bo Luo, Feng Luo, Vasile Palade, and M. Arif Wani, editors, *16th IEEE International Conference on Machine Learning and Applications, ICMLA 2017, Cancun, Mexico, December 18-21, 2017*, pages 737–742. IEEE, 2017.

[42] A. Bergamini de Neira, A. Medeiros Araujo, and M. Nogueira. Early botnet detection for the internet and the internet of things by autonomous machine learning. In *16th International Conference on Mobility, Sensing and Networking, MSN 2020, Tokyo, Japan, December 17-19, 2020*, pages 516–523. IEEE, 2020.

[43] John W. Dorfinger P., Panholzer G. Real-time detection of encrypted traffic based on entropy estimation. Master's thesis, 2011.

[44] Dheeru Dua and Casey Graff. UCI Machine Learning Repository. `http://archive.ics.uci.edu/ml`, 2017. Accessed: 15-Oct-2021.

[45] M. A.Hall E. Frank and I. H.Witten. *The WEKA workbench.* MK, 2016. Accessed: 10-Nov-2021.

[46] Claude E.Shannon. Prediction and entropy of printed english. *Bell system technical journal*, 30:50–64, January 1951.

[47] Tyrell Fawcett. Exfild: a tool for the detection of data exfiltration using entropy and encryption characteristics of network traffic. Master's thesis, University of Delaware, 2010.

[48] L. Ferreira, A. Luiz Pilastri, C. Manuel Martins, P. Miguel Pires, and P. Cortez. A comparison of automl tools for machine learning, deep learning and xgboost. In *International Joint Conference on Neural Networks, IJCNN 2021, Shenzhen, China, July 18-22, 2021*, pages 1–8. IEEE, 2021.

[49] R. Gandhi. Naive Bayes Classifier. `https://towardsdatascience.com/naive-bayes-classifier-81d512f50a7c`, 2018. Accessed: Nov-2021.

[50] F. Haddadi and N. Zincir-Heywood. Benchmarking the effect of flow exporters and protocol filters on botnet traffic classification. *IEEE Syst. J.*, 10(4):1390–1401, 2016.

[51] J. Hale. TPOT Automated Machine Learning in Python. `https://towardsdatascience.com/tpot-automated-machine-learning-in-python-4c063b3e5de9`, 2018. Accessed: 1-Nov-2021.

[52] D. Hjelm. A New Needle and Haystack: Detecting DNS over HTTPS Usage. `https://www.sans.org/reading-room/whitepapers/dns/needle-haystack-detecting-dns-https-usage-39160`. Accessed: 10-May-2021.

[53] Arash Habibi Lashkari Iman Sharafaldin and Ali A. Ghorbani. CIC-IDS 2017. `https://www.unb.ca/cic/datasets/ids-2017.html`. Accessed: 5-Mar-2021.

[54] J.Ahmed, H. Habibi Gharakheili, Q. Raza, C. Russell, and V. Sivaraman. Real-time detection of DNS exfiltration and tunneling from enterprise networks. In Joe Betser, Carol J. Fung, Alex Clemm, Jérôme François, and Shingo Ata, editors, *IFIP/IEEE International Symposium on Integrated Network Management, IM 2019, Washington, DC, USA, April 09-11, 2019*, pages 649–653. IFIP, 2019.

[55] S. Khanchi, A. Vahdat, M. I. Heywood, and Nur Zincir-Heywood. On botnet detection with genetic programming under streaming data label budgets and class imbalance. *Swarm Evol. Comput.*, 39:123–140, 2018.

[56] Nilima Khanna. J48 Classification (C4.5 Algorithm) in a Nutshell. `https://medium.com/@nilimakhanna1/j48-classification-c4-5-algorithm-in-a-nutshell-24c50d20658e`, 2021. Accessed: 18-Nov-2021.

[57] Y. Khodjaeva and N. Zincir-Heywood. Network flow entropy for identifying malicious behaviours in DNS tunnels. In Delphine Reinhardt and Tilo Müller, editors, *ARES 2021: The 16th International Conference on Availability, Reliability and Security, Vienna, Austria, August 17-20, 2021*, pages 72:1–72:7. ACM, 2021.

[58] A. Khormali, J. Park, H. Alasmary, A. Anwar, M. Saad, and D. A. Mohaisen. Domain name system security and privacy: A contemporary survey. *Comput. Networks*, 185:107699, 2021.

[59] Kh.Shahbar and N. Zincir-Heywood. How far can we push flow analysis to identify encrypted anonymity network traffic? In *2018 IEEE/IFIP Network Operations and Management Symposium, NOMS 2018, Taipei, Taiwan, April 23-27, 2018*, pages 1–6. IEEE, 2018.

[60] A. Lakhina, M. Crovella, and Ch. Diot. Mining anomalies using traffic feature distributions. In R. Guérin, R. Govindan, and G. Minshall, editors, *Proceedings of the ACM SIGCOMM 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Philadelphia, Pennsylvania, USA, August 22-26, 2005*, pages 217–228. ACM, 2005.

[61] D. Lambion, M. Josten, F. G. Olumofin, and M. De Cock. Malicious DNS tunneling detection in real-traffic DNS data. In X. Wu, Ch. Jermaine, L. Xiong, X. Hu, O. Kotevska, S. Lu, W. Xu, S. Aluru, C. Zhai, E. Al-Masri, Zh. Chen, and J. Saltz, editors, *2020 IEEE International Conference on Big Data (IEEE BigData 2020), Atlanta, GA, USA, December 10-13, 2020*, pages 5736–5738. IEEE, 2020.

[62] Duc C. Le and N. Zincir-Heywood. A frontier: Dependable, reliable and secure machine learning for network/system management. *J. Netw. Syst. Manag.*, 28(4):827–849, 2020.

[63] Duc C. Le, N. Zincir-Heywood, and M. I. Heywood. Data analytics on network traffic flows for botnet behaviour detection. In *2016 IEEE Symposium Series on Computational Intelligence, SSCI 2016, Athens, Greece, December 6-9, 2016*, pages 1–7. IEEE, 2016.

[64] Ch. Lu, B. Liu, Zh. Li, Sh. Hao, H. Duan, M. Zhang, Ch. Leng, Y. Liu, Z. Zhang, and J. Wu. An end-to-end, large-scale measurement of dns-over-encryption: How far have we come? In *Proceedings of the Internet Measurement Conference, IMC 2019, Amsterdam, The Netherlands, October 21-23, 2019*, pages 22–35. ACM, 2019.

[65] M. Nidhal Mejri and J. Ben-Othman. Entropy as a new metric for denial of service attack detection in vehicular ad-hoc networks. In Ravi Prakash, Azzedine Boukerche, Cheng Li, and Falko Dressler, editors, *17th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems, MSWiM'14, Montreal, QC, Canada, September 21-26, 2014*, pages 73–79. ACM, 2014.

[66] M.MontazeriShatoori, L. Davidson, G. Kaur, and A. Habibi Lashkari. Detection of doh tunnels using time-series classification of encrypted traffic. In *IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress, DASC/PiCom/CBDCom/CyberSciTech 2020, Calgary, AB, Canada, August 17-22, 2020*, pages 63–70. IEEE, 2020.

[67] R. S. Olson, N. Bartley, R. J. Urbanowicz, and J. H. Moore. Evaluation of a tree-based pipeline optimization tool for automating data science. In Tobias Friedrich, Frank Neumann, and Andrew M. Sutton, editors, *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference, Denver, CO, USA, July 20 - 24, 2016*, pages 485–492. ACM, 2016.

[68] F. Pacheco, E. Exposito, M. Gineste, C. Baudoin, and J. Aguilar. Towards the deployment of machine learning solutions in network traffic classification: A systematic survey. *IEEE Commun. Surv. Tutorials*, 21(2):1988–2014, 2019.

[69] S. Patel. Chapter 2: SVM (Support Vector Machines) - Theory. `https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72`, 2017. Accessed: Nov-2021.

[70] R. Raghav, Pratheesh, K. Shedbalkar, Minal Moharir, N Deepamala, P Ramakanth Kumar, and MGP Tanmayananda. Analysis and detection of malicious activity on doh traffic. In *2021 2nd Global Conference for Advancement in Technology (GCAT)*, pages 1–5, 2021.

[71] S. Ray. 6 Easy Steps to Learn Naive Bayes Algorithm with codes in Python and R. `https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/`, 2017. Accessed: Nov-2021.

[72] Madison Schott. Random Forest Algorithm for Machine Learning. `https://medium.com/capital-one-tech/random-forest-algorithm-for-machine-learning-c4b2c8cc9feb`, 2019. Accessed: 17-Nov-2021.

[73] M. Seufert, R. Schatz, N. Wehner, B. Gardlo, and P. Casas. Is QUIC becoming the new tcp? on the potential impact of a new protocol on networked multimedia qoe. In *11th International Conference on Quality of Multimedia Experience QoMEX 2019, Berlin, Germany, June 5-7, 2019*, pages 1–6. IEEE, 2019.

[74] I. Sharafaldin, A. Habibi Lashkari, and A. A. Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In Paolo Mori, Steven Furnell, and Olivier Camp, editors, *Proceedings of the 4th International Conference on Information Systems Security and Privacy, ICISSP 2018, Funchal, Madeira - Portugal, January 22-24, 2018*, pages 108–116. SciTePress, 2018.

[75] S. Kumar Singh and P. Kumar Roy. Detecting malicious dns over https traffic using machine learning. 2020.

[76] B. Stecanella. Support Vector Machines (SVM) Algorithms Explained. `https://monkeylearn.com/blog/introduction-to-support-vector-machines-svm/`, 2017. Accessed: Nov-2021.

[77] Georgia Tech. GT Malware Passive DNS Data Daily Feed. `http://dx.doi.org/10.23721/102/1354027`. Accessed: 6-Mar-2021.

[78] G. Vormayr, J. Fabini, and T. Zseby. Why are my flows different? A tutorial on flow exporters. *IEEE Commun. Surv. Tutorials*, 22(3):2064–2103, 2020.

[79] M. Zhou, Sh. Zhang, Y. Qiu, H. Luo, and Zh. Wu. Entropy-based spammer detection. In *Proceedings of the 10th International Conference on Internet Multimedia Computing and Service, Nanjing, China, August 17-19, 2018*, pages 43:1–43:6. ACM, 2018.