Nicolas Kyejo

Implementation of a Forensic Analysis System for Malicious Network Traffic

Metropolia University of Applied Sciences

Bachelor of Engineering

Information and Communication Technology

Bachelor's Thesis

1 May 2023

Abstract

Author:	Nicolas Kyejo
Title:	Implementation of a forensic analysis system for malicious network traffic
Number of Pages:	34 pages
Date:	1 May 2023
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communication Technology
Professional Major:	IoT and Cloud Computing
Supervisors:	Janne Salonen, Head of School (ICT)

The objective of this thesis was to develop and evaluate a forensic system for detecting malicious network traffic, focusing on its practical implementation and effectiveness as a low-cost security solution. To support this goal, only open and freely available tools were used.

The main solution used in the thesis project was the creation of machine learning models to detect malicious network traffic. These models were created from K-nearest Neighbors, Logistic Regression, Random Forest, and Multi-Layer Perceptron algorithms. The dataset used in the training of the models was the CICIDS2017 dataset. As a final evaluation step, traffic captured in a virtual LAN was used the assess the models.

The performance and predictions of the models indicated that they could be used effectively in a network forensic system for identifying cyberattacks. The thesis showed that such an implemented system was profoundly reliant on the availability of open datasets, hence the cost in terms of effort seems to be justified if quality and open datasets are available and used.

Keywords: network forensics, packets, intrusion detection, machine-learning, cybersecurity

Tiivistelmä

Tekijä:	Nicolas Kyejo
Otsikko:	Haitallisen verkkoliikenteen rikosteknisen analyysijärjestelmän toteuttaminen
Sivumäärä:	34 sivua
Aika:	1.5.2023
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	IoT ja Pilvipalvelut
Ohjaajat:	Janne Salonen, Osaamisaluepäällikkö, ICT ja tuotantotalous

Insinöörityön päätavoitteena oli koneoppimismallien luominen pahantahtoisen verkkoliikenteen havaitsemiseksi. Nämä mallit luotiin käyttäen ‘K-nearest Neighbors’-, ‘Logistic Regression’-, ‘Random Forest’-, ja ‘Multi-Layer Perceptron’ -algoritmeja. Mallien koulutuksessa käytetty datasetti oli CICIDS2017-datasetti. Lopullisessa arviointivaiheessa virtuaalisessa lähiverkossa kaapattua liikennettä käytettiin mallien arvioimiseen.

Mallien suorituskyky ja ennusteet viittasivat siihen, että niitä voitaisiin käyttää tehokkaasti verkkorikostekniikassa kyberhyökkäysten tunnistamiseksi. Insinöörityö osoitti, että tällainen toteutettu järjestelmä oli erittäin riippuvainen avoimien datasettien saatavuudesta, joten vaiva näyttää olevan perusteltua, jos laadukkaita avoimia datasetteja on saatavilla ja käytössä.

Avainsanat: tunkeutumisen tunnistusjärjestelmä, koneoppiminen, kyberturvallisuus

Contents

List of Abbreviations

AIDS: Anomaly-based Intrusion Detection System.

ANN: Artificial Neural Network.

DDoS: Distributed Denial-of-Service.

DNS: Domain Name System.

DOM: Document Object Model.

DoS: Denial-of-Service.

DVWA: Damn Vulnerable Web Application.

FTP: File Transfer Protocol.

GUI: Graphical User Interface.

HTTP: Hypertext Transfer Protocol.

ICMP: Internet Control Message Protocol.

IDS: Intrusion Detection System.

IPS: Intrusion Prevention System.

KNN: K-nearest Neighbors.

LAN: Local Area Network.

MLP: Multi-Layer Perceptron.

PCAP: Packet Capture.

SIDS: Signature-based Intrusion Detection System.

SIEM: Security Information and Event Management.

SQL: Structured Query Language.

SSH: Secure Shell.

XSS: Cross-Site Scripting.

Glossary

cross-validation: a method to test the accuracy of a machine learning model by splitting the dataset into smaller subsets, training the model on

some subsets and testing it on the others. This process is repeated multiple times to ensure the model's performance is not overfitting and to estimate its generalization performance on new data.

decision threshold: a probability value that is used to classify observations into one of two (or more) classes. If the predicted probability of an observation belonging to one class is above the decision threshold, it is classified as belonging to that class; otherwise, it is classified as belonging to the other class.

ensemble method: a technique that combines the predictions of multiple learning algorithms to improve prediction.

false-negative: a missed positive prediction.

false-positive: an incorrect classification of a negative prediction as positive, that is to say, noise.

feature extraction: a process of selecting and transforming raw data into a set of meaningful features that can be used for model training and prediction.

hyperparameters: parameters set before training in a machine learning model and are not learned from the data. They control the behavior of the learning algorithm and can significantly impact the performance of the trained model.

Linear Regression: a statistical method used to model the relationship between two variables by fitting a straight line through the data points to minimize the sum of squared residuals, with the goal of predicting the value of one variable (dependent variable) based on the value of another variable (independent variable).

overfitting: a situation where a trained model performs very well on the training data but fails to generalize to new, unseen data.

promiscuous mode: a feature in packet capture where a network interface card captures all packets on a network, including those not addressed to it.

test data: a separate and independent dataset that is used to evaluate the performance and generalization capability of a trained machine learning model.

train data: a labeled dataset used to train a machine learning model.

zero-day: a type of vulnerability or flaw in software, hardware, or firmware known only to the attacker and not to the developers of the affected system, making it difficult to defend against.

1 Introduction

In the world of Information Technology, cyberattacks are quite ubiquitous. A cursory look at the news might even suggest that we should expect a service or a device we use to come under a successful attack if it has not already been so. It therefore becomes quite important to determine with a certain confidence that a cyberattack took place.

The main goal of this thesis is to investigate how a forensic system for detecting malicious network traffic could be implemented in practice. In this research, the aim is to evaluate such a system's effectiveness and whether it could be useful as a low-cost forensic security solution. To achieve this goal, the thesis will first briefly analyze the current state of tools available for detecting malicious traffic. Furthermore, some background concerning malicious network traffic and supervised machine learning methods will be explored and discussed. The thesis will then finally examine the system implemented, and draw conclusions from the implementation result.

The thesis contains a limited scope—it does not compare the implemented system with existing paid and free solutions for which there are many. Moreover, it does not take into account usage of the forensic system in an active real-world network. Therefore, the effectiveness of the system is speculated since the generation of malicious network traffic is in a controlled setting.

2 Current State Analysis

The field of IT is very important to the modern economic infrastructure since it helps facilitate nearly all modern commerce, logistics, business, research, healthcare, and more. Its crucial role has made it a highly valuable target for cybercriminals hoping to gain money and influence. As stated previously, having a way to analyze and demonstrate with a degree of certainty that a cyberattack took place is one of the goals in forensic analysis of network traffic.

In the current technology tools offering, there are some tools that can determine whether a cyberattack took place. These tools are usually classed as Security Information and Event Management (SIEM), Intrusion Prevention System (IPS), and Intrusion Detection System (IDS).

SIEM tools are usually mostly employed as a ‘Software as a service’ (SaaS) solution in a Cloud platform to solve security and compliance requirements as required by specific industries. Examples of such compliance standards include the Sarbanes-Oxley Act (SOX), General Data Protection Regulation (GDPR), and others. SIEMs are good at determining and alerting the presence of cyberattacks; however, they are a fairly expensive investment for small organizations or individuals with smaller needs in terms of data traffic volume and functionality.

On the other hand, IPSes and IDSes are less expensive in comparison to SIEM solutions. They are employed as either an on-premise or a Cloud solution in the form of hardware or software to detect cyberattacks, and in the case of IPS also prevent detected cyberattacks. Hardware-based IPSes and IDSes are generally faster and more expensive compared to software-based counterparts.

The aforementioned tools have something in common—they require a non-trivial investment in money and resources. Therefore, this thesis will examine whether building a similar system is practical and achievable in terms of effort and cost. It will do this by evaluating a software-based forensic system that was built by

making use of available open datasets to train machine learning models to identify whether a particular piece of network traffic is malicious or benign. The implemented system is not a drop-in replacement for the mentioned tools, rather it is a proof of concept of building an ‘IDS-like’ functionality with only open data and freely available tools.

3 Theoretical background

The following sections will detail relevant background concerning malicious network traffic, IDS workings, and supervised machine learning methods. The chosen types of malicious network traffic and their descriptions will be detailed. Furthermore, a brief explanation of IDS, its classes and methods of detection will be described. In the same manner, a few selected machine learning algorithms that are useful in classification problems will be briefly explained.

3.1 Malicious Network Traffic

Malicious network traffic is any network traffic that is intended to harm or breach information systems without the consent of their owners. As an example, The United States of America's Computer Fraud and Abuse Act (CFAA) bill includes this in a more expansive definition to include cause of damage in monetary form, loss of data, modification of data, extortion, physical, death or otherwise [1]. General examples of malicious network traffic include Denial-of-Service (DoS) attacks, phishing, malware delivery, and ransomware among others. In the context of this thesis, the following cyberattacks are considered in the implemented solution: code injection specifically Structured Query Language (SQL) injection, command injection, Cross-Site Scripting (XSS), DoS attacks, and brute force attacks.

3.1.1 SQL Injection

Code injection is a method to input code in an application that was not meant to be executed by that application [2]. As a form of attack it means inputting code that is malicious to be executed by an application. In a non-attack form, this means adding code that is benign, for instance, adding some extra functionality that is not present in an application. SQL injection is a type of code injection where SQL statements are passed and executed by an application that builds and executes SQL queries from input passed by a user. A trivial example of an SQL injection

can be an e-commerce application that contains a form where a user can search for products; expected input from the application's point of view is input such as "SD card" or "M.2 SSD".

```
1 SELECT * FROM items where items.name = 'SD card'; --normal
      query
2 SELECT * FROM items WHERE items.name = 'SD card'; DROP table
      items--' --query with injected code
```

Listing 1: SQL statements showing an SQL injection

Assuming in this example that the data is fetched from an SQL database, the application will build an SQL query from the user input as shown in listing 1 line number one. Without proper input validation in the application, one could pass input such as "SD card'; DROP table items--" resulting in the built query shown in listing 1 line number two. The executed query will cause the particular database table to be deleted.

3.1.2 Command Injection

Another type of code injection is command injection. This differs from SQL injection in that the extra parsed code is executed by the underlying OS. Both types of injection are caused by improper validation of user input.

An example of command injection can be an online note application that prior to initializing a writing environment asks for the file name to use; assuming no validation of user input is done, an attacker could pass extra commands to the underlying OS to be executed, as in `touch filename ; cat /etc/passwd`. The extra statement is parsed after the semicolon which indicates that what follows is a sequential command to run. This could potentially expose sensitive information or grant unauthorized access to the attacker when other commands are passed.

3.1.3 Cross-Site Scripting Attack

Similarly, XSS is a type of code injection that allows an attacker to inject malicious code into a vulnerable web application. It is able to bypass the same origin policy

which controls the access of data by and from different domains. This attack can compromise user data and allow an attacker to perform actions as the compromised-user. [3]

There are three main types of XSS attacks, namely: reflected, stored, and Document Object Model (DOM)-based XSS. Reflected XSS occurs when an attacker injects malicious code into a web request URL, which is then “reflected” back to the user when they access the URL. Stored XSS, on the other hand, is when the malicious code is stored on the server by an attacker, waiting for unsuspecting users to visit the compromised page. Finally, DOM-based XSS is a type of XSS that targets the DOM of a web page. This type of attack is possible when a web application relies on processing client-side JavaScript to manipulate the DOM. If the DOM can be manipulated by an attacker in this way, they can include malicious code to be run as part of the processing. [3]

3.1.4 Denial-of-Service Attack

A DoS attack is an attack that aims to disrupt the normal functioning of a device or an application service. The most common case of a DoS attack is the disruption of a web service by sending it a flood of requests such that it is unable to contend with the number of requests, resulting in the service being unavailable or unable to respond to further requests. It is important to note that a DoS attack is launched from a single machine while a Distributed Denial-of-Service (DDoS), a more advanced form of the same attack, is launched from multiple machines which makes it more difficult to mitigate against. [4]

According to Cloudflare, a DoS attack is categorized as either a buffer overflow attack or a flood attack [4]. In this thesis, an Internet Control Message Protocol (ICMP) ping flood DoS attack is used in the implementation of the attack system. An ICMP ping flood attack works by sending multiple ICMP echo requests packets to a target server—if a server is not configured to mitigate against this type of attack, it will reply to each ICMP echo request packet with a ICMP echo reply packet thereby consuming resources proportional to the number of requests it

received [5]. When the number of requests overwhelms the server's resources, it will result in a DoS.

3.1.5 Brute Force Attack

Brute force attack in cybersecurity is a type of attack where an attacker's method of gaining access to a system is to guess and try different combination of passwords through trial-and-error. In this method the attacker is limited by time, method of guessing, processing power (in case the attack is offline), and possibly any mitigations in the target system.

Due to the advent of powerful GPU hardware technology, the ease of cracking weak passwords has increased dramatically. If an attacker has access to a hashed password list through any means, they can crack passwords in just a few days with a mid-range GPU. The suggested mitigation against brute force attacks is to use longer passwords, stronger hashing functions such as bCrypt (a popular hashing function) and perhaps usage of different password schemes such as biometric and graphical passwords. [6]

3.2 Intrusion Detection System

An IDS is a software or hardware system designed to identify and alert on any unauthorized activities that could cause harm to an information system [7]. They can be broadly categorized into two groups: Signature-based Intrusion Detection System (SIDS) and Anomaly-based Intrusion Detection System (AIDS). They can also be classified by division into host-based IDS, network-based IDS, and hybrid-based IDS [8]; this latter classification is based on the where the source of data used for analysis is collected from, more specifically, either from individual devices, a network segment such as a Local Area Network (LAN), or a combination of these.

A SIDS works by having a database of known malicious signatures and it uses those to detect intrusion from collected network traffic or host logs [9; 8; 7]. Due to

its method of detection, it can only detect previous flagged signatures and therefore faces significant difficulties in detecting unknown intrusion signatures especially zero-day attacks [7].

On the other hand, AIDS works by defining ‘normal’ behavior of a network and/or computer system by the use of knowledge-based, statistical-based, or machine learning methods; any meaningful deviation from the defined normal behavior is assumed to be an intrusion. [7]. One advantage of AIDS compared to SIDS is that it can detect previously unseen attacks [8][9]—however, this ability can lead do higher rate of false positive detections due to the threshold of separating malicious and benign behavior [7].

3.3 Supervised Machine Learning Methods

Supervised machine learning is a type of machine learning that involves training a model on labeled data to predict the output for new, unseen data. In supervised learning, the model is trained using a dataset that contains inputs and their corresponding correct outputs, also known as labels or targets. The goal of the model is then to learn a mapping function between the input and the output, which can then be used for later predictions. [10]

As mentioned previously, the supporting goal of the thesis is to detect whether a particular set of captured traffic is malicious or not; to achieve this goal, a set of supervised machine learning methods that can be used in classification problems were chosen, namely—Logistic Regression, K-nearest Neighbors (KNN), Random Forest and Multi-Layer Perceptron (MLP). Note that the selected machine learning methods do not constitute all possible methods that can be used to solve classification problems.

3.3.1 K-nearest Neighbors

KNN is a non-parametric and lazy learning algorithm that classifies a data point based on its proximity to other data points in a training set. In this context ‘lazy

learning' means it does not try to learn a general mapping function between inputs and outputs during the training phase, but instead stores the entire training dataset in memory and waits until a new data point is presented to predict it.

Various distance metrics exist to calculate this proximity such as Euclidean distance, Manhattan distance, Hamming distance, and others. The most popular distance metric is the Euclidean distance which is given by the formula $\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$ where x and y are respective variables (Eucledian vectors) of data points. [11]

An example of how KNN works visually can be seen in figure 1. The green datapoint is the datapoint that is going to be classified as either belonging to red or blue.

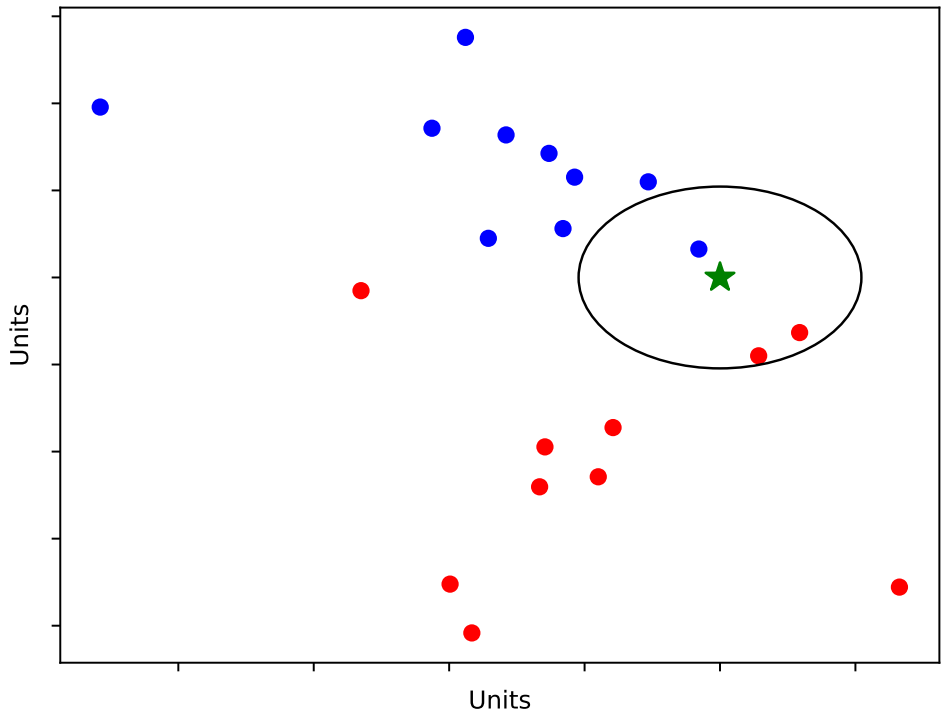

Figure 1: KNN algorithm with a random dataset where the K value is three

The figure 1 shows a scatter plot with red and blue labels. The KNN algorithm works by calculating the distance metric (Euclidean distance in this case) between

the new data point and rest of the data points stored in memory. From the nearest distance points chosen (three in this case), the label for the new data point is determined to be red if the points are considered to have the same weight (uniform). On the other hand, the label would be blue if we give closer data points a higher weight than those far away.

Most of the work in KNN involves choosing the value of K with the help of cross-validation to determine the value of K that results in a good test score comparatively. The base performance of KNN is determined by the choice of K value, distance metric, and feature scaling. [11]

3.3.2 Logistic Regression

Another supervised machine learning method used in classification tasks is Logistic Regression. It predicts the probability of an input instance belonging to one of two possible classes by applying an activation function to the output of Linear Regression prediction.

During a Logistic Regression's model training process, Linear Regression is used to estimate the coefficients that provide the best fit for the input data with respect to the output data; a sigmoid function—the most common used activation function in Logistic Regression is then used to estimate the probability range between zero and one [12]. Additional activation functions include ReLU, tanh, softmax, and others. An example of a sigmoid function is shown in figure 2.

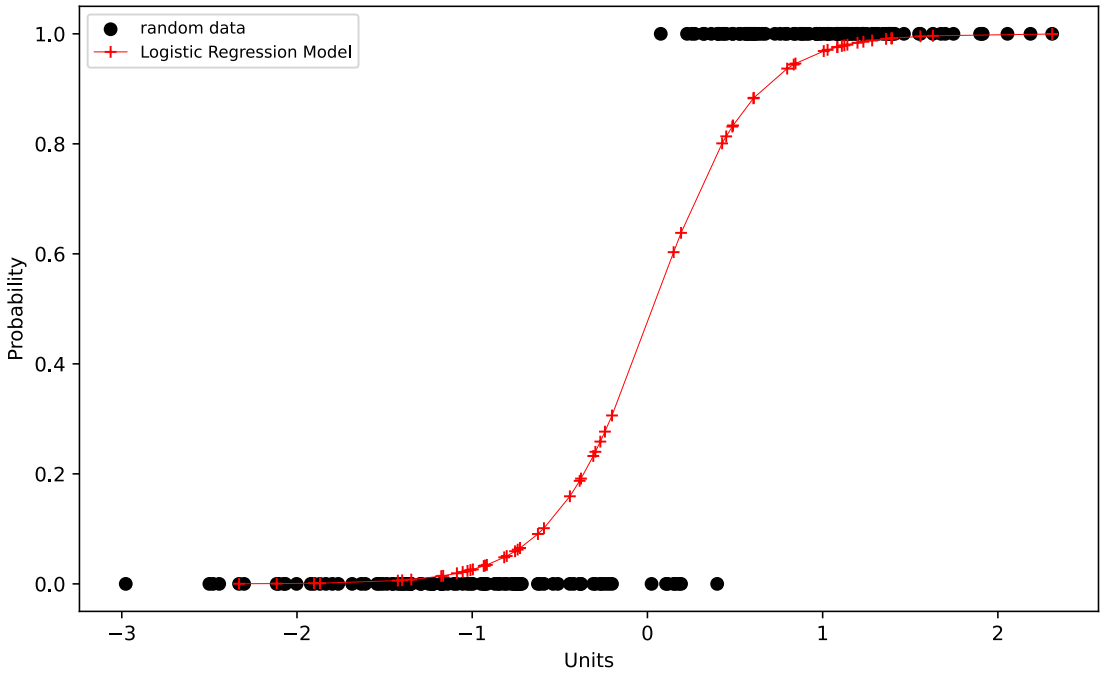

Figure 2: An example of logistic regression with a random dataset

The random dataset in figure 2 shows a prediction curve for a portion of the dataset set aside as test data; the random data already contains labels zero and one, hence they lie exactly at zero and one in the figure. For the test data, a prediction is made after training a Logistic Regression model on the train data. In a binary classification problem, the decision threshold can be chosen to be 0.5 as in this case, therefore if the probability prediction is greater than 0.5 the output is classified as one and vice-versa.

3.3.3 Random Forest

Random Forest is an ensemble method that makes use of multiple uncorrelated decision trees generated through bagging and feature randomness to make predictions. In classification tasks, random forests output a prediction based on the majority class encountered in the forest. [13]

An example of Random Forest used in a multi-class (more than two) classification is shown in figure 3. In this example, there are four decision trees in the forest.

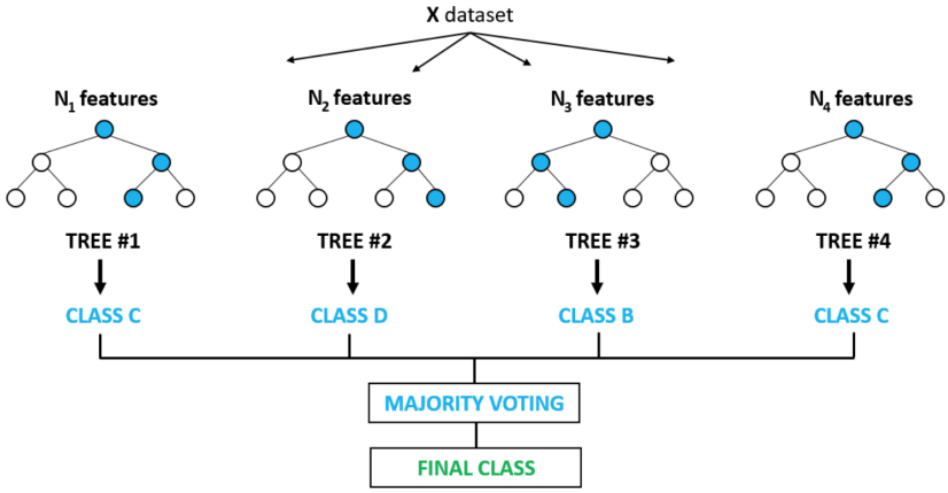

Figure 3: A Random Forest classification example (Copied from [14])

In the figure 3, the prediction through majority voting is *class C*. The main benefit purported by Random Forest algorithms include reduced risk of overfitting [13] and resistance to redundant variables [14]; however, they tend to be complex in terms of interpretability [13].

3.3.4 Multi-layer Perceptron

Finally, a MLP is a type of Artificial Neural Network (ANN) with a feedforward mechanism (outputs are forwarded to the next layer) characterized by an architecture that consists of an input layer, hidden layer(s), and an output layer; it additionally makes use of backpropagation to adjust the weights of neurons to minimize the cost function of the output(s). [15].

An example of a MLP ANN can be seen in figure 4 with three neurons in the input layer, two neurons in one hidden layer, and one neuron in the output layer.

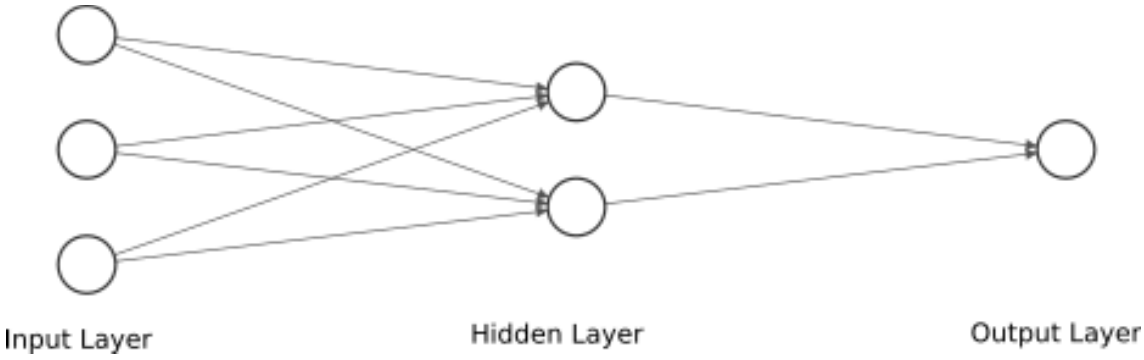

Figure 4: A simple MLP ANN with one hidden layer

In the figure 4, the output of the two neurons in the hidden layer are a result of the application of an activation function. Likewise the output of the neuron in the output layer is also activated—the activation functions used need not be the same in different layers. Factors such as the number of hidden layers, number of iterations (steps to reduce the cost function), momentum and learning rate have an effect on the performance of a MLP model [15].

4 Implementation

The following sections detail the implemented solution to flag network traffic as malicious or benign. The criteria for selecting tools used in the implementation was that they be freely available and open-source. The implemented solution was divided into two separate parts, namely the network environment setup and model training parts.

The network environment setup dealt with generating malicious and benign traffic in a virtualized environment to be used later in testing the prediction of the models trained. This part was not strictly necessary as traffic can be generated and captured in any network environment; the rationale of creating such an environment was to capture only needed traffic in a safe environment, to keep the size of capture packets small, and to increase privacy as captured packets might contain other network data when capturing in promiscuous mode.

The model training part was concerned with creating machine learning models to analyze and flag the captured traffic in the virtual network. Additionally, it also dealt with choosing the dataset used in training the models and other details involved in the whole process.

4.1 Network Environment Setup

The virtualized network environment was created with the help of *Vagrant*; *Vagrant* is a tool that can be used to quickly provision virtual machines. The environment created with *Vagrant* consisted of three machines all in the same network for simplicity since the aim was to generate and capture traffic. All machines were created using a base image of Kali Linux, which is a popular OS used commonly in security testing scenarios. The names and roles were assigned were as follows:

- *defender*: to capture traffic in the network
- *attacker*: to generate malicious traffic
- *target*: to act as a vulnerable server that can be attacked.

The *target* machine had a version of Damn Vulnerable Web Application (DVWA) running on it. DVWA is an intentionally insecure web application which makes it a good candidate for testing malicious attacks. The vulnerabilities discussed in chapter 3.1 are all possible in DVWA.

```
config.vm.define "vulnerable-target" do |target|
  target.vm.hostname = "target"
  target.vm.network "private_network", ip: "192.168.60.60"
  target.vm.provider :virtualbox do |vb|
    vb.gui = false
    vb.name = "target"
    vb.memory = 1024
    vb.cpus = 1
  end
  target.vm.provision "bootstrap", type: "shell", path:
      "./make_dvwa_accessible_to_lan.sh", run: "once"
  target.vm.provision "start-dvwa", type: "shell", inline: "sudo
      dvwa-start", run: "always"
end
```

Listing 2: Provisioning the *target* machine with *Vagrant*

The listing 2 shows the code used in creating the *target* machine. While it is possible to use other providers, *Virtualbox* was chosen because it had support in *Vagrant* to modify network device settings which was needed to set promiscuous mode in the *defender* machine.

With the virtual LAN created, the network traffic was generated in both malicious and benign stages. In the malicious stage, the *attacker* generated traffic for DoS, SQL injection, command injection, brute force, and XSS attacks. The *attacker* machine in figure 5 shows how malicious network traffic was generated in the case of SQL injection that was aimed at the *target* machine.

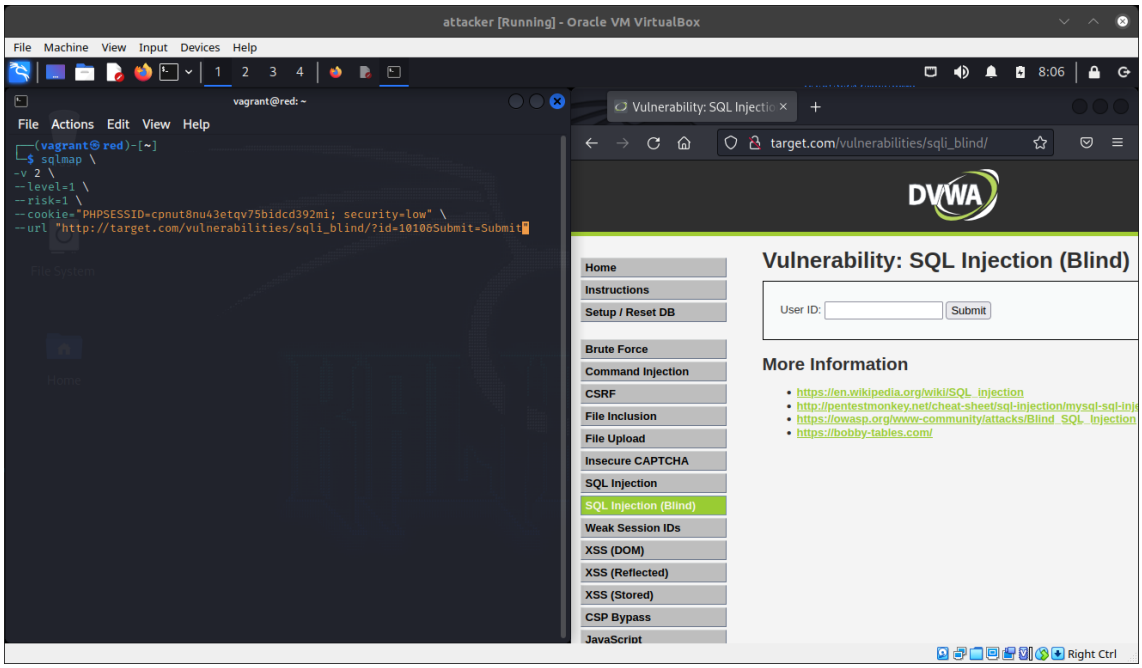

Figure 5: Screenshot of *attacker* machine preparing an SQL injection test

The figure 5 shows in view the web application to be attacked and the program *sqlmap* to automate discovery of SQL vulnerabilities. In contrast to the malicious stage which consisted only of traffic captured in the virtual LAN, the benign stage consisted of both virtual LAN and internet directed traffic. The benign traffic consisted of web browsing of the *target* machine's web application via Hypertext Transfer Protocol (HTTP) without any attacks. The rest of the benign traffic consisted of Telnet, Domain Name System (DNS), Secure Shell (SSH), and File Transfer Protocol (FTP) protocol-related traffic. Each scenario of the malicious and benign stages was captured in Packet Capture (PCAP) file format by the *defender* machine using *tcpdump*. As a final step, the benign PCAP files were renamed to include the word 'benign' in the filename to distinguish them from the malicious captures.

4.2 Model Training

This section focuses on the model training process and how it was implemented with machine learning methods: KNN, Logistic Regression, MLP, and Random Forest. Training the models involved feeding them with labeled data to learn the

patterns and relationships within the data. The training was performed on a laptop with an ‘AMD® Ryzen 7 PRO 4750U’ processor and 30.6 GiB of SODIMM DDR4 memory.

4.2.1 Dataset

The training process for classification tasks needs labeled data for supervised machine learning. The quality, quantity, and representativeness of the dataset used can greatly impact the performance of the trained models. It was deemed that the labeled data should contain at least some features that can be extracted (feature extraction) easily from PCAP files.

The dataset eventually selected for use was the CICIDS2017 intrusion detection evaluation dataset [16]. A newer version did exist at the start of the implementation (CSE-CIC-IDS2018), nonetheless CICIDS2017 was used due to its relative small size (under 900 MB). The CICIDS2017 dataset with labeled data existed in CSV file format, those were downloaded and used in the model training. Figure 6 visually shows the composition of the CICIDS2017 dataset.

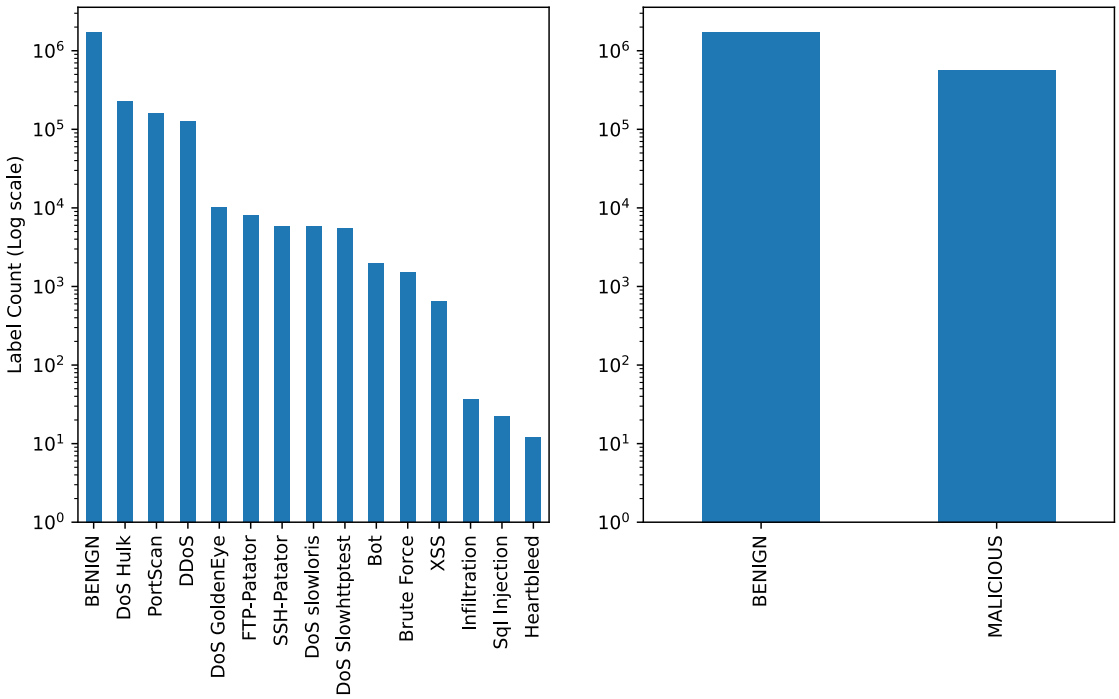

Figure 6: Dataset label distribution

The dataset label distribution can be observed in figure 6 showing the proportion of malicious labels to benign labels. Due to the proportion of benign labels being higher in comparison to malicious labels (2,271,320 versus 556,556), one CSV file from the dataset (`Monday-WorkingHours.pcap_ISCX.csv`) was dropped from the training process since it contained only benign labels. The dataset was further preprocessed by removing identified redundant columns and one duplicate column; moreover rows with infinity numbers were removed the dataset. Finally, the columns with malicious labels were converted to the class number one while benign labels were converted to the class number zero to make the training process straight-forward. In the end the combined dataset after preprocessing had 2,298,395 rows with 71 columns where seventy columns were for features and one column for labels.

4.2.2 Training Process

The model training part as stated previously involves finding patterns within the dataset to make predictions for whether a particular piece of network traffic is malicious. In the implementation of this stage, the *scikit-learn* [17] library was chosen due to the availability of documentation and examples; another reason was that it did not have any dependencies to GPU libraries, making it easy to install on different OS platforms. The use of a machine learning library significantly reduced the implementation time since no time would be spent on algorithm coding. The steps involved in the training process consisted of:

- Creating the model pipeline.
- Training and tuning of hyperparameters.
- Cross-validation of models.
- Saving models in a persistent format.

The model pipeline is a list of steps that are chained together and can be applied to a piece of data input. Listing 3 shows a KNN model pipeline as used in the training process implementation.

```python
def knn_model() -> Pipeline:
    return make_pipeline(
        StandardScaler(),
        LinearDiscriminantAnalysis(),
        KNeighborsClassifier(
            n_neighbors=3,
            p=2,
        ),
        verbose=True,
    )
```

Listing 3: Setup of KNN pipeline

The pipeline shown in listing 3 contains three steps: scaling, dimensionality reduction, and the K-neighbors classifier itself. The scaling of features step is an additional preprocessing step which depending on the learning algorithm itself might be needed or not; in addition, the dimensionality reduction step helps reduce computation time—here it reduces the data dimensions from 70 to 1. The last step in the pipeline is the KNN classifier with $K = 3$ and $p = 2$ to use the Euclidean distance metric (see chapter 3.3.1). Contrast the parameters in listing 3 with listing 4.

```python
    return make_pipeline(
        RandomForestClassifier(
            n_estimators=100,
            criterion="gini",
            max_depth=None,
            max_features=35,
            min_samples_split=2,
            min_samples_leaf=2,
            bootstrap=False,
            max_samples=None,
            random_state=SEED,
            verbose=1,
            class_weight="balanced",
        ),
        verbose=True,
    )
```

Listing 4: Setup of Random Forest pipeline

As can be noted in listing 4, some models can have more tunable hyperparameters. Another difference that can be observed is that Random Forest does not need feature scaling since the algorithm is not sensitive to unscaled features. Furthermore, the pipeline contains only one step so essentially the classifier can be used directly without including it in a pipeline; the reason it was included is to make it easily comparable to the other models and additionally for code type checking.

The training part is straight-forward, the dataset is split into train data and test data, the portion of the test data was chosen to be a third of the dataset. Each time the model pipeline was ran, the test score (calculated on the test data) and time taken was noted. By tuning the hyperparameters, varying test scores could be observed; in this way, a specific model's hyperparameters that perform best were eventually used. Note that, the hyperparameters that had good performance where not necessarily used in the final training, the time to train the model was also taken into account. Therefore a balance between the two was the deciding factor on which hyperparameters were used. For training the final models with the parameters chosen—the whole dataset is used, there was no need to set aside train data.

Be aware that there exist software libraries to automate the tuning of hyperparameters to find the best performing parameters or even the best predictive model itself. This path was abandoned after a few rounds due to unreliability problems where the process would crash after a few days of training.

Additionally, cross-validation of the models was performed. This was done to ensure that the models were not overfitting on the train data. The cross-validation strategy chosen was a stratified k-fold wherein the train data was split into five folds and the test score accuracy was compared. A stratified strategy in the cross-validation was crucial since the dataset labels were not in equal proportions (see figure 6).

The last step in the training process was saving the models trained. The importance of this step is that there is no need to train the model again in order to use it in network traffic prediction—this saves time and computation resources. The other rationale is if sharing of the model is needed later. There exist several formats for saving models, each with their advantage and disadvantages, for this implementation, the *skops* library was chosen. The main reason *skops* was chosen is that it was created to work with the *scikit-learn* library which made it easy to integrate within the project. One disadvantage however with *skops* is that the models trained cannot be imported in other programming languages, therefore the saved models can only work with the specific version of Python programming language and *scikit-learn* library used in the training process.

4.2.3 Prediction Process

Before the prediction based on the saved models was done, the PCAP files were first converted to CSV files; this was done with the help of a software project called *cicflowmeter*. The version available at the time the project started was forked and modified to fix some errors and consequently included in this project to do the conversion part. The importance of the conversion from PCAP to CSV is that feature extraction is performed on the network traffic data. The feature extraction process itself is out of scope of this thesis. However, it suffices to say that *cicflowmeter* (Python implementation) is based on *CICFlowMeter* which is a Java software associated with the CICIDS2017 dataset used for feature extraction, among its other uses.

The prediction process in the implementation consisted of predicting the collected network traffic from the LAN. For quick predictions, a Graphical User Interface (GUI) was created to facilitate this process as show in figure 7.

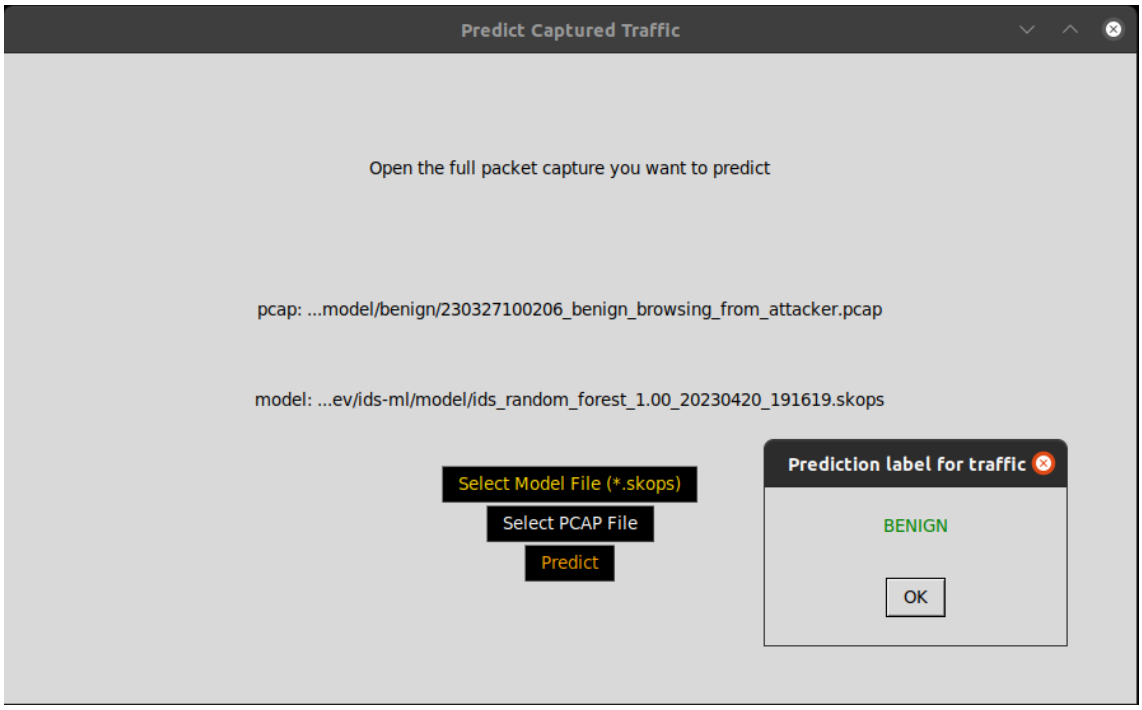

Figure 7: Screenshot of the simple prediction GUI

The GUI in figure 7 was created through Python's interface to the Tcl/Tk GUI toolkit called *tkinter*. The choice for using *tkinter* instead of other more newer GUI toolkits, for instance, the *Qt* framework, was simply because the functionality desired was basic, namely, a few buttons, and a way to open files.

Through testing prediction of different models over time, it became cumbersome to observe and record predictions in this way. As an alternative, a script to run predictions of different models on multiple PCAP files was implemented; the script was meant to create a simple report showing how correct the predictions were in reality. Figure 8 shows how the report looked in practice.

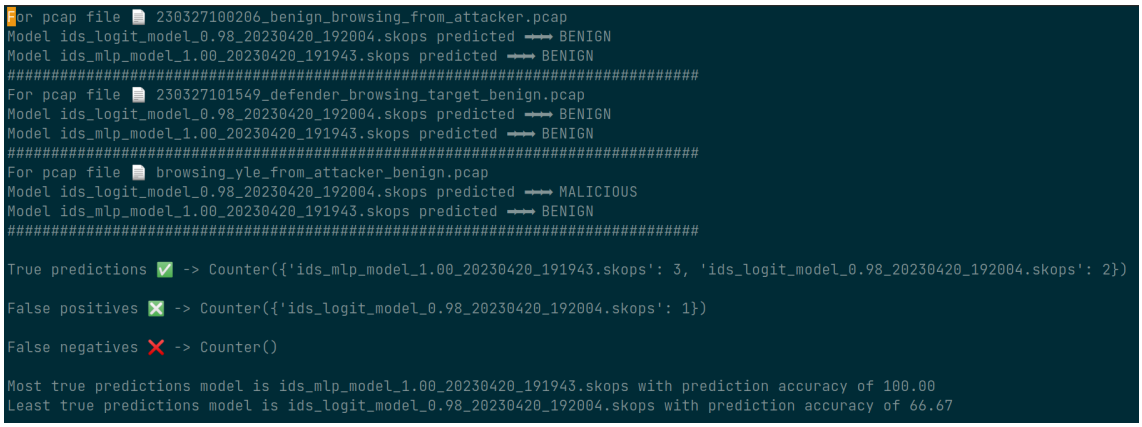

Figure 8: Screenshot of the simple report content

The figure 8 also shows that the simple report created by the script details the count of true predictions, false-positives and false-negatives. Both the GUI and the script methods were used in different scenarios depending on what effort was required in regards to whether a single prediction was desired or multiple.

The full source code for the solution implementation can be found at https://github.com/nicolaskyejo/ids-ml-exploration, it contains both the network environment setup and the model hyperparameters used for the results obtained in the following chapter.

5 Results

This chapter provides an evaluation of the models in terms of performance and the prediction results obtained. The first section, Model Performance, describes the accuracy of the models and their ability to generalize to unseen data. It compares and analyzes the respective performance on the CICIDS2017 dataset. The second section, Prediction Results, presents the results of applying the models to data that was gathered from the implemented LAN. It provides an analysis of the predictions made by the models, including the accuracy and reliability of the results. Overall, this chapter provides insights into the performance and effectiveness of the models trained.

Recall, precision, and F1-score are evaluation metrics commonly used in binary classification tasks. Recall measures the proportion of actual positive samples that are correctly identified as positive by the model while precision measures the proportion of predicted positive samples that are actually positive. F1-score is the harmonic mean of recall and precision, which takes both metrics into account and provides an evaluation of a model's performance. The F1-score ranges from zero to one, where one represents perfect precision and recall, and zero represents the worst performance. The formula 1 shows the calculation of the F1-score.

$$\text{F1} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{1}$$

where precision and recall are:

$$\text{precision} = \frac{TP}{TP + FP} \tag{2}$$

$$\text{recall} = \frac{TP}{TP + FN} \tag{3}$$

In the formulas 2 and 3, TP represents the number of true positives, FP represents the number of false-positives, and FN represents the number of false-negatives. True positive in the implemented system represents the case where the network traffic is malicious. In the *scikit-learn* library, scores are calculated by passing the score metric that is desired and it will be calculated automatically. These metrics are essential in evaluating the performance of a binary classification model and can help identify the strengths and weaknesses of the model in distinguishing between the two classes.

5.1 Model Performance

The evaluation of a model's base performance was calculated on the basis of the dataset, whereas the collected network traffic was used in the final prediction. For each model, different evaluations were done and recorded. In this section, these evaluations are compared with each other.

One of the most important predictors of a model's performance is the learning curve. A learning curve is a graphical representation that illustrates the progress of a model's learning by plotting its accuracy on the train data and test data against the number of training samples. The graph in figure 9 shows the learning curve of the chosen machine learning models which depicts this relationship.

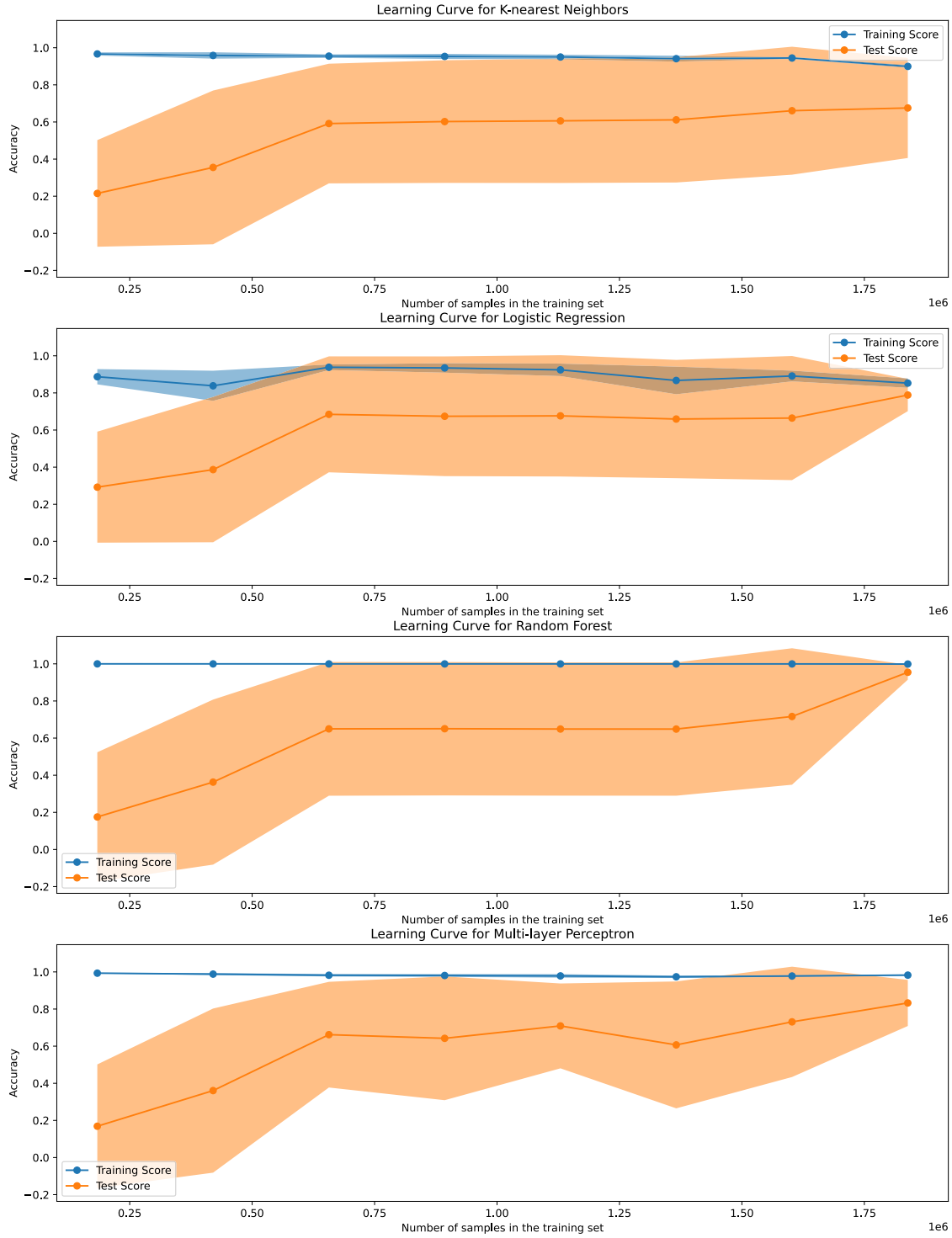

Figure 9: Learning curve of the models on the dataset

As shown in figure 9, the models have different curves signifying that each model improved at different rates when more training samples were added. The training score evaluates how the models performed on the train data, in other words, how

well it was able to generalize the relationship between the input and output when the answers were known. All models improved with the addition of more samples in the training set up to a point—this point was somewhere at about 36% of the whole dataset samples. Also of note is how Random Forest had a plateau while the other models except KNN slightly degraded in test score accuracy after this point. A slightly more numerical comparison can be seen in figure 10.

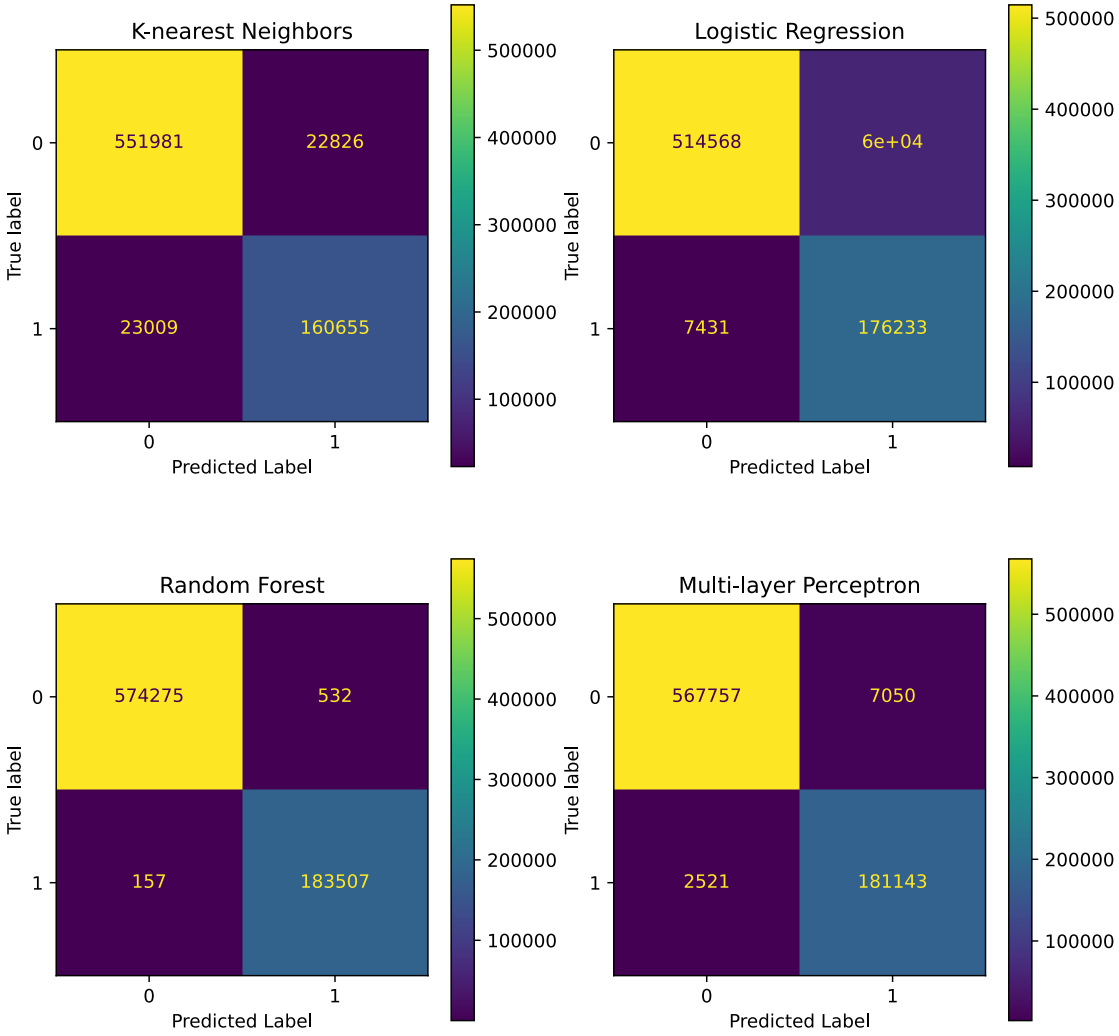

Figure 10: Confusion matrix of the models

The figure 10 shows the confusion matrix diagram. The number of TP, FP, FN, and TN can be gleamed from the matrix. It can be seen that the Random Forest model had the highest true predictions and the Logistic Regression model the highest missed predictions. A more intuitive comparison can be observed in figure 11 showing the mean F1-score of five cross-validated scores.

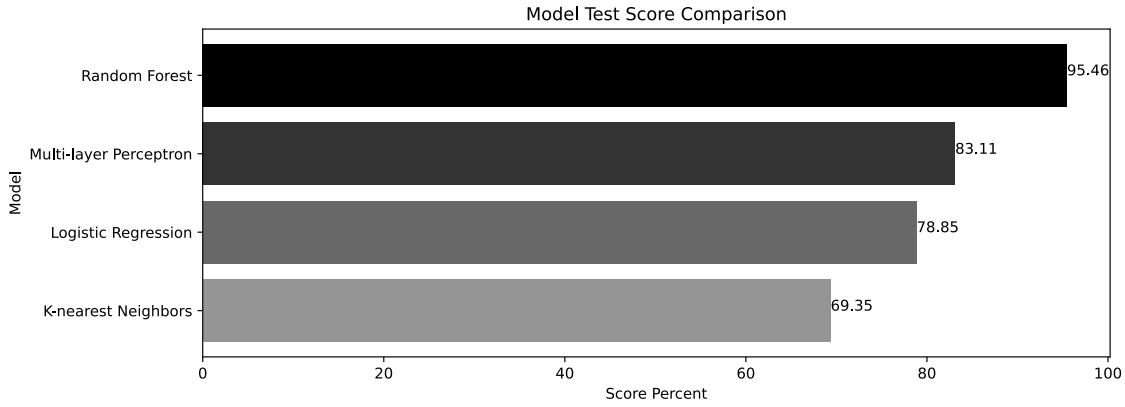

Figure 11: Mean F1-score of models

The information learned from figure 11 is similar to figure 10. It can be seen clearly that Random Forest had the highest score and KNN the lowest. An explanation of why KNN score is the lowest is that the F1-score does not take into account the number of TN.

Another useful comparison from the training process was how the models performed in terms of time taken to train; this information can affect resource planning optimization when considered together with model accuracy. This comparison can be seen in figure 12.

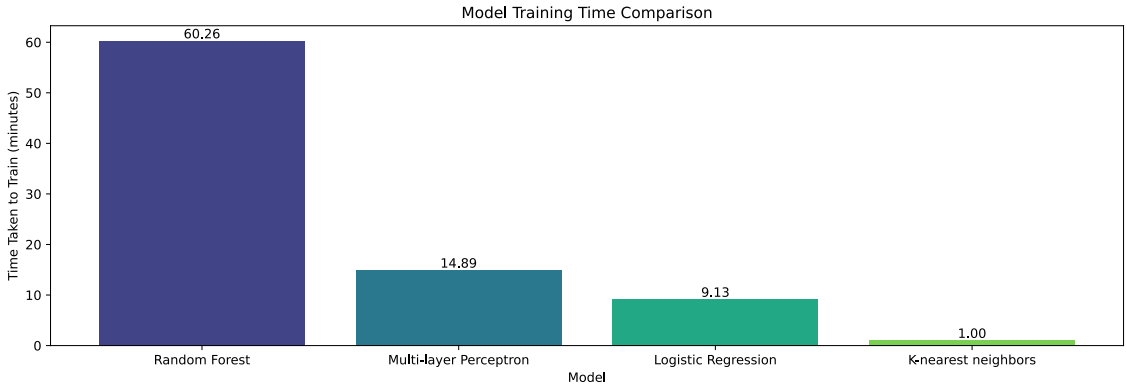

Figure 12: Model training time on full dataset

The training time shown in figure 12 reveals that Random Forest took the longest time to train while KNN the shortest. In regards to KNN, the result is not surprising since the model's computation only takes place during prediction of new data. The training process results show that all the models used can be justified as good models when their training time and F1-scores are examined together.

5.2 Prediction Results

In this section, the prediction results obtained from the trained models on the captured network data (PCAP) are discussed. The prediction results are analyzed to evaluate the performance of each model in terms of precision, recall, and F1-score. By comparing the results of different models, the most effective model for predicting network traffic can be identified. Additionally, the potential implications of the prediction results are briefly addressed. Table 1 shows the predictions of the captured network traffic in the LAN.

Table 1: Prediction results for different models on captured network data

PCAP type	**Label**	**KNN**	**MLP**	**RF**	**LR**
Web browsing on target	Benign	Benign	Malicious	Benign	Benign
Telnet connection	Benign	Benign	Benign	Benign	Benign
FTP connection	Benign	Benign	Benign	Benign	Benign
SSH connection	Benign	Benign	Benign	Benign	Benign
DNS query	Benign	Benign	Benign	Benign	Benign
Brute force login	Malicious	Malicious	Malicious	Benign	Benign
DoS by ICMP flood	Malicious	Malicious	Benign	Benign	Malicious
SQL injection	Malicious	Malicious	Malicious	Benign	Malicious
Command injection	Malicious	Malicious	Malicious	Benign	Malicious
XSS	Malicious	Benign	Malicious	Benign	Benign

From the data in table 1, the results suggest that KNN is the most accurate model. The F1-scores (see formula 1) of KNN, MLP, Random Forest, and Logistic Regression are 0.89, 0.80, 0.00, and 0.75 respectively. The expectation from the final model performance is somewhat different—especially in regards to Random Forest and KNN. One explanation could be that Random Forest was overfitting on the dataset and KNN model did not. KNN as the most accurate model in the final results predicted everything correctly except the XSS attack; MLP ANN as the second most accurate model incorrectly flagged HTTP web browsing traffic as malicious and missed the DoS attack. The Logistic Regression model as the third most accurate model missed the brute force and XSS attacks.

According to the dataset publication, command injection attack was not part of the dataset. Despite this, the models were able to correctly flag it as malicious (except Random Forest) which can be evidence of the power of AIDS to detect new or unseen attacks.

However, it should be noted that to calculate a meaningful F1-score, more observation data is needed with varying attack and benign samples. Therefore, the final results are only a suggestive result rather than a conclusive result. In the end, a combination of AIDS to detect zero-day attacks and SIDS to detect known attacks may be the most effective method in forensics of malicious network traffic.

6 Conclusion

The primary goals of this thesis were to detect malicious network traffic, investigate the practical implementation of a forensic system for this purpose, and evaluate its effectiveness as a low-cost security solution. Through the application of supervised machine learning methods, it was demonstrated that it is possible to detect malicious traffic in a reliable way.

Regarding the practical implementation process, it was heavily reliant on the availability of open datasets. Without available open datasets, the process would have been arduous. The training process itself was fairly straight-forward with most time spent on tuning hyperparameters. Therefore, the goal of practical implementation is indeed realistic to achieve when quality datasets are obtainable.

As to the effectiveness as a low-cost security solution, it depends on the resources available to an individual or an organization. For entities with considerable capital and resources, it can be feasible to allocate some of those resources to train such network forensic models and even create datasets that are used for that purpose. However, in a market economy it can be seen as a waste of resources to do so, therefore, it is more probable that a few organizations or companies sell their own solutions to others. Nevertheless, it can be assumed that organizations with high-security profiles, for example, government agencies and military facilities are creating and using their own network forensic solutions which are considered as a low-cost security solution to them.

While this thesis has achieved its main goal, it is important to recognize some limitations as well. One weakness is the limited the number of attack type scenarios that were examined, both in the actual dataset and the ones generated for the final evaluation; more attack data would have raised the confidence of the final results. Another limitation is that the trained models were not tested in a real-world network which could have brought interesting observations.

In closing, one area of further research recommended is the development of open datasets that can be used to create network forensic solutions. Most of the current available datasets rely on creating malicious traffic in a virtualized environment and labelling them appropriately; this can be considered a cumbersome and time-consuming process. It could be feasible to crowdsource the creation of such datasets with multiple organizations contributing their traffic captures and logs. The inherent privacy issues of such an approach could be alleviated with some form of anonymization techniques, perhaps again with the help of machine learning methods.

References

1 18 § 1030 - ”Fraud and Related Activity in Connection with Computers” 2020. Cornell Law School. <https://www.law.cornell.edu/uscode/text/18/1030>. Accessed on 18 February 2023.

2 Mitropoulos, Dimitris; Karakoidas, Vassilios; Louridas, Panagiotis & Spinellis, Diomidis. Jan. 2011. “Countering code injection attacks: a unified approach”. In: Information Management & Computer Security 19.3, pp. 177–194. <https://doi.org/10.1108/09685221111153555>.

3 Cross-site scripting (XSS) 2023. PortSwigger Ltd. Online. <https://portswigger.net/web-security/cross-site-scripting>. Accessed on 6 April 2023.

4 What is a denial-of-service (DoS) attack? 2023. Cloudflare, Inc. Online. <https://www.cloudflare.com/learning/ddos/glossary/denial-of-service/>. Accessed on 16 February 2023.

5 Ping (ICMP) flood DDoS attack 2023. Cloudflare, Inc. Online. <https://www.cloudflare.com/learning/ddos/ping-icmp-flood-ddos-attack/>. Accessed on 16 February 2023.

6 Bošnjak, L.; Sreš, J. & Brumen, B. 2018. “Brute-force and dictionary attack on hashed real-world passwords”. In: *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pp. 1161–1166.

7 Khraisat, Ansam; Gondal, Iqbal; Vamplew, Peter & Kamruzzaman, Joarder. 2019. “Survey of intrusion detection systems: techniques, datasets and challenges”. In: Cybersecurity 2.1, pp. 1–22.

8 Rajasekaran, K & Nirmala, K. 2012. “Classification and importance of intrusion detection system”. In: International Journal of Computer Science and Information Security 10.8, p. 44.

9 Soniya, S. Sobin & Vigila, S. Maria Celestin. 2016. “Intrusion detection system: Classification and techniques”. In: *2016 International Conference on Circuit, Power and Computing Technologies (ICCPCT)*, pp. 1–7.

10 Jiang, Tammy; Gradus, Jaimie L. & Rosellini, Anthony J. 2020. “Supervised Machine Learning: A Brief Primer”. In: Behavior Therapy 51.5, pp. 675–687. <https://www.sciencedirect.com/science/article/pii/S0005789420300678>.

11 Taunk, Kashvi; De, Sanjukta; Verma, Srishti & Swetapadma, Aleena. 2019. “A Brief Review of Nearest Neighbor Algorithm for Learning and

Classification”. In: *2019 International Conference on Intelligent Computing and Control Systems (ICCS)*, pp. 1255–1260.

12 Zou, Xiaonan; Hu, Yong; Tian, Zhewen & Shen, Kaiyuan. 2019. “Logistic Regression Model Optimization and Case Analysis”. In: *2019 IEEE 7th International Conference on Computer Science and Network Technology (ICCSNT)*, pp. 135–139.

13 What is random forest? 2023. IBM, Corp. Online. <https://www.ibm.com/topics/random-forest>. Accessed on 18 April 2023.

14 Kirasich, Kaitlin; Smith, Trace & Sadler, Bivin. 2018. “Random forest vs logistic regression: binary classification for heterogeneous datasets”. In: SMU Data Science Review 1.3, p. 9.

15 Mas, JF. 2018. “Multilayer perceptron (MLP)”. In: Geomatic approaches for modeling land change scenarios, pp. 451–455.

16 Sharafaldin, Iman; Lashkari, Arash Habibi & Ghorbani, Ali A. 2018. “Toward generating a new intrusion detection dataset and intrusion traffic characterization.” In: ICISSp 1, pp. 108–116.

17 Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; Vanderplas, J.; Passos, A.; Cournapeau, D.; Brucher, M.; Perrot, M. & Duchesnay, E. 2011. “Scikit-learn: Machine Learning in Python”. In: Journal of Machine Learning Research 12, pp. 2825–2830.