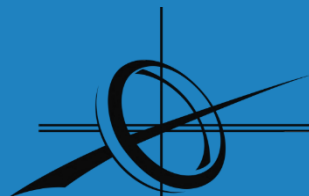




POLITÉCNICA



Universidad
Politécnica
de Madrid

**ETSI SISTEMAS
INFORMÁTICOS**

Diseño e implementación de un sistema de detección y gestión de Inyección de Denegación de Servicio en redes atacadas aplicando técnicas de inteligencia artificial

Final Degree Project

Degree in Information Society Technologies

Author:

David Ramos Archilla

Tutor:

Borja Bordel Sánchez

February 2024

UNIVERSIDAD TÉCNICA DE MADRID ESCUELA
TÉCNICA SUPERIOR DE INGENIERÍA DE SISTEMAS
Nº



**Diseño e implementación de un
sistema de detección y gestión de
Denegación de Denegación de
Denegación de Denegación de
Servicio en redes ataques aplicando técnicas de inteligencia artificial**

Final Degree Project

Degree in Society Technologies

Curso académico 2023-2024

Author:

David Ramos Archilla

Tutor:

Borja Bordel Sánchez

I would like to take this opportunity to express my most sincere thanks to the people who have been instrumental in the completion of my final degree project. 'on. ~This moment would not have been possible without the support and encouragement of those who have ~~af~~le in this academic journey. First of all, I would like to deeply thank my family, thanks to them I have reached where I am today. 'hI would like to thank my tutor for his support, who has guided and helped me throughout 'on the project.

Summary

This work consists of the construction of a system that acts as a firewall for the detection of denial of service attacks. To perform these detections, Machine Learning and Deep Learning techniques have been used to train models with these capabilities. In addition, a graphical interface has been implemented through which the traffic received by the system can be observed in real time, as well as the predictions made by the system as to whether the traffic is an attack or not.

First of all, in order to train an artificial intelligence model, training data is needed. A dataset has been chosen that contains different characteristics about a large number of network flows and a label indicating whether it is an attack, and if so, the type of attack it is. After choosing the dataset, it is necessary to perform a preprocessing stage, which consists of preparing the data in order to have a good training phase. Once the data is ready, two models have been trained. The first one is a neural network, specifically a multilayer perceptron, and the second one is a gradient boosting algorithm based on decision trees. After training, evaluating and comparing them, it was obtained that the gradient boosting model outperforms the neural network, achieving an accuracy of 99 % on the validation data set, so it was this model that was used to start up the system.

On the other hand, it has been necessary to implement a tool to capture the network traffic, and which is also capable of obtaining the same characteristics of the network flows used for training the models. For this purpose, an existing tool has been used as a basis, developed by the same university that created the dataset used. However, this tool does not have all the necessary requirements to be included in this work, so we have implemented a version of it that does.

The graphic interface that allows visualising the traffic has been implemented and all of this has been integrated under the same system. After carrying out several experiments, the results obtained are quite promising, as the system has been able to detect denial of service attacks.

Abstract

This work consists of building a system that acts as a firewall for detecting denial-of-service attacks. To perform these detections, Machine Learning and Deep Learning techniques have been employed to train models with these capabilities. In addition, a graphical interface has been implemented through which the traffic received by the system can be observed in real-time, as well as the predictions it makes about whether or not the traffic is an attack.

Firstly, to train an artificial intelligence model, training data is required. A dataset has been chosen that contains different features about a large number of network flows and a label indicating whether it is an attack, and if so, what type of attack it is. After choosing the dataset, a preprocessing stage is necessary, which consists of preparing the data so that the training phase is good. Once the data is ready, two models have been trained. The first is a neural network, specifically, a multilayer perceptron, and the second is a gradient boosting algorithm based on decision trees. After training, evaluating, and comparing them, it has been found that the gradient boosting model outperforms the neural network, achieving 99 % accuracy on the validation dataset, so this model has been used when launching the system.

On the other hand, it has been necessary to implement a tool to capture network traffic, which is also capable of obtaining the same features about network flows that are used for model training. To do this, an existing tool developed by the same university that created the dataset used has been used as a base. However, this tool does not meet all the requirements necessary to be included in this work, so a version of it that does meet the requirements has been implemented.

Finally, a graphical interface has been implemented that allows traffic to be visualised and all of this has been integrated under the same system. After conducting several experiments, quite promising results have been obtained, as the system has been able to detect denial-of-service attacks.

Índex

Acknowledgements	I
Summary	II
Abstract	III
1. Introduction	1
1.1. Context	1
1.2. Objectives	1
1.3. Structure of the document	2
2. State of the art	3
2.1. Distributed Denial of Service (DDoS) attacks	3
2.1.1. TCP Attacks	4
2.1.2. UDP Attacks	5
2.1.3. Attacks with reflexi�on MS SQL	5
2.1.4. Attacks with reflexi�on LDAP	5
2.2. Artificial intelligence	6
2.2.1. Machine Learning	6
2.2.2. Deep Learning	8
2.3. Tools and libraries	11
2.4. Similar works	12
3. Project development	13
3.1. Dataset	13
3.1.1. Processing of the dataset	13
3.1.2. Analysis of the dataset	14
3.2. Model training	16
3.2.1. Cross-validation	17
3.2.2. Neural network	17
3.2.3. Gradient boosting model	23
3.3. Traffic capture tool	27
3.3.1. PyFlowmeter	27
3.4. System architecture	28
3.4.1. Server	29
3.4.2. Artificial intelligence model	30
3.4.3. Web interface	30
4. Results	33
4.1. System in operation	33
4.1.1. Tr�afico in real time	34
4.1.2. LDAP attack	35
4.1.3. UDP Attack null	36
4.1.4. TCP SYN flood attack	37
5. Conclusions and future work	38
5.1. Conclusions	38
5.2. Social and environmental impact	38
5.3. Future lines	39

<i>INDEX</i>	V
--------------	---

Bibliography	40
---------------------	-----------

Annexes	43
----------------	-----------

A. C project source code	44
A.1. preprocessed dataset.py	44
A.2. neuralnet.py	45
A.3. search hyperparameters.py	46
A.4. pcap to csv pyflowmeter.py	47
A.5. send real traffic to server.py	47
A.6. simulate traffic pyflowmeter.py	47
A.7. endpoint start sniffer.py	48
A.8. send traffic and get data.py	49
A.9. router.js	49
A.10. firewall model.py	50

Index of tables

2.1. Main activation functions on for neural networks.....	10
3.1. Report classification on neural network during cross-validation	20
3.2. Final neural network on classification report	21
3.3. Gradient boosting model classification report on.....	25

Índice of figures

2.1. Diagram of a DDoS attack.....	3
2.2. Schematic diagram of a SYN flood attack on SYN.....	4
2.3. Diagram of an attack with reflexi on LDAP.....	5
2.4. Artificial intelligence concept map.....	6
2.5. Phases of automatic learning atic learning.....	7
2.6. Basic operation of gradient boosting algorithms	8
2.7. Comparison on between Machine Learning and Deep Learning.....	9
2.8. Image of a neural network	10
3.1. Visualisation on of the dataset	14
3.2. Distribution on of types of attacks	15
3.3. Distribution of attack types after balancing on	16
3.4. Cross-validation scheme	17
3.5. Gr áfica of error during cross-validation in neural network	21
3.6. Gr áfica of error during training of the final neural network	22
3.7. Confounding matrix on of the final neural network	22
3.8. Importance of hyperpar ámetros	24
3.9. Hist órica accuracy on during optimisation on.....	24
3.10. Distribution on of the parameters tested during the optimisation on on	24
3.11. Confounding matrix on of the gradient boosting model.....	26
3.12. Importance of input variables	26
3.13. System diagram	28
3.14. Dashboard view	31
3.15. Vista traffic-analysis	32
4.1. Table of the analysis of the real-time traffic received table of analysis	34
4.2. Gr áficas del an álysis del tr álysis del tr áfico recibido en tiempo real.....	34
4.3. Table of the analysis of an LDAP attack álysis of an LDAP attack	35
4.4. Gr áficas del an álysis de un ataque LDAP.....	35
4.5. Table of the analysis of a null UDP attack	36
4.6. Gr áficas del an álysis de un ataque UDP null.....	36
4.7. Table of analysis of a TCP SYN flood attack.....	37
4.8. Gr áficas del an álysis de un ataque TCP SYN flood.....	37

Chapter 1



1.1. Context

In today's digital era, the increasing reliance on online services has led to greater vulnerability to various forms of cyber-attacks. Among these threats, distributed denial of service (DDoS) attacks stand out as a strategy aimed at disrupting access to online services such as websites. This phenomenon is manifested by the saturation of malicious traffic from sources, with the purpose of exceeding the resource capacity of the targeted server.

A report by Cloudflare [1], hypervolumetric DDoS attacks have increased over the last few years. In the first quarter of 2023, a significant increase in the number of hypervolumetric DDoS attacks was observed, the largest of which reached a peak of over 71 million requests per second, 55 % above the previous world record of 46 million requests per second by Google. In addition, Cisco's annual Internet report [2], the number of distributed denial of service attacks has gradually increased in the last few years. In 2018, 7.9 million attacks were recorded, in 2019 it will increase to 9.5 million and in 2020 it will rise to 10.8 million.

On the other hand, the advances in the field of artificial intelligence in the last few years have been enormous, and a large number of new applications have emerged that build on these advances. The core of these advances has been Machine Learning, a branch of artificial intelligence that focuses on the development of algorithms and models that allow machines to learn patterns and make decisions without explicit programming, improving their performance with experience and data. Within Machine Learning is Deep Learning, a discipline that uses deep neural networks as learning models to understand and process data and sometimes perform more complex tasks. An example of the application of these new artificial intelligence techniques has been in the detection of denial of service attacks.

This is the reason why this work proposes the use of Machine Learning and Deep Learning techniques to try to detect distributed denial of service attacks from the traffic received.

1.2. Objectives

In this work we propose to create a system that captures network packets, analyses them using an artificial intelligence model to detect possible denial of service attacks and then displays them on an interface.

on containing additional information on the traffic. With this in mind, the following objectives are defined:

- Train a Machine Learning model that can detect denial of service attacks and classify them the type of attack they are. The model should provide the confidence with which it makes each prediction.
- Find and implement a system that captures the network traffic in real time and is able to extract variables about the traffic.
- Find and implement a graphical interface through which to observe the network traffic and the predictions made by the Machine Learning model on it.
- Integrate the Machine Learning model, the traffic capture tool and the interface on a server.

1.3. Structure of the document

The following is a description of the structure of this document:

- Chapter 1 serves as a brief introduction to the work, explaining the context in which it is carried out and the objectives set.
- Chapter 2 gives an overview of the state of the art and technologies that have been used during the development of the work.
- Chapter 3 provides a detailed explanation of the process carried out in the development of the work.
- Chapter 4 shows results that have been obtained after the work has been carried out.
- Chapter 5 discusses the conclusions drawn from the work, as well as possible future lines of research.

Chapter 2

State of the art

2.1. Attacks of denial of service attacks on attacks Distributed Denial of Service (DDoS) attacks

Distributed denial of service (DDoS) attacks are a type of cyber-attack that aims to disrupt the normal operation of the targeted service or network by consuming its available bandwidth, exhausting its computational resources or causing other types of failures. As a result, legitimate users are unable to access the attacked resource, resulting in a denial of service for them.

This is typically done using botnets, which are computers that have been infected with malware and can be remotely controlled without their owners being aware of it. These 'zombie' computers send continuous and massive requests to the victim, which causes a significant increase in network traffic and can cause the victim's system to crash or become slow and difficult to access. The fact that the attacks are distributed, i.e. they are carried out from a large number of machines, makes them very difficult to detect and mitigate, as the attack traffic is barely indistinguishable from legitimate traffic.

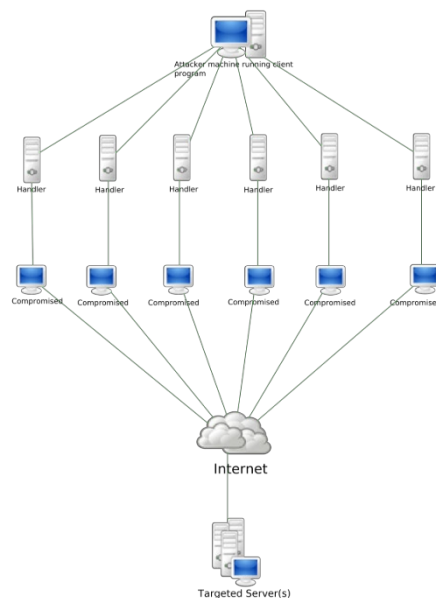


Figure 2.1: Schematic diagram of a DDoS attack

2.1. DISTRIBUTED DENIAL OF SERVICE ATTACKS (DDOS) 4

One of the largest DDoS attacks was suffered by the company Github in 2018, which took just over as long as 10 minutes to detect. The attack sent to GitHub's servers 1.3 terabits per second of traffic, sent at a rate of 126.9 million packets per second. In 2020, Amazon Web Services (AWS) reported to successfully mitigate a 2.3 terabits per second distributed denial of service attack, marking a milestone as the largest DDoS attack engagement in history to date. In the AWS report [3], this incident involved a DDoS reflection attack using CLDAP. Finally, on 1 June 2022, a Google Cloud Armor customer was hit with a DDoS attack using HTTPS that reached a maximum of 46 million requests per second. To put this into perspective, this is equivalent to receiving all the requests Wikipedia receives in a day but in just 10 seconds [4].

There are a large number of different types of DDoS attacks, although most share some common characteristics. One of the most common classifications is based on the protocol layer against which they target such as the transport layer, the network layer or the application layer. In the remainder of this section, a description of the types of attacks against which the models have been trained will be given.

2.1.1. TCP attacks

Within TCP (Transmission Control Protocol) attacks, the most common form of attack is known as SYN flooding (SYN flooding). These attacks take advantage of how a new connection is established in the TCP protocol. First, to initiate the connection, the client sends a SYN packet to the server. Then, the server replies with a SYN/ACK packet. Finally, the client sends back an ACK packet to acknowledge receipt of the packet from the server.

With this in mind, to perform a denial-of-service attack, an attacker sends a high volume of SYN packets to a server. The server would then respond to each of these packets, leaving ports open waiting to receive the ACK packets. The key part of the attack is that clients would never send ACK packets, leaving the server waiting.

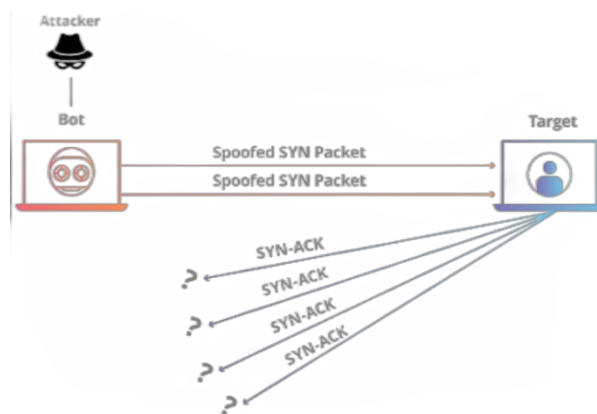


Figure 2.2: Diagram of a SYN flooding attack on SYN

2.1.2. UDP attacks

This type of attack works in a similar way to SYN flooding attacks on TCP, but in this case it is performed on the User Datagram Protocol (UDP). In this situation, when a server receives a UDP packet on a given port, it first checks if there is a program running on listening for requests on that port. If there is, the server responds with an ICMP (ping) packet to inform that the destination cannot be reached.

To saturate the server, the attacker sends a large number of UDP packets, so that the server must devote its resources to check and respond to every UDP packet received. Thus, the server resources can be exhausted very quickly if the attack is sufficiently volumetric.

2.1.3. Attacks with ~~an~~ MS SQL

This type of attack exploits vulnerabilities in Microsoft SQL Server databases to amplify and escalate the attacks. The attack occurs using the Microsoft SQL Server Privilege Protocol (MC-SQLR), which is used whenever a client needs to obtain information from MS SQL Server. When connecting to a database server, the client receives a list of database instances in response. This is where attackers send spoofed requests that appear to come from the victim, in order to make the vulnerable server reply to the target with packets much larger than the original ones, generating an amplification of the traffic to the victim.

2.1.4. Attacks with LDAP

Attacks with LDAP (Lightweight Directory Access Protocol) reflection work in a very similar way to attacks with MSSQL reflection. The difference is that in this case the requests with the spoofed IP address are made to LDAP servers, which, like SQL servers, amplify the traffic as their responses are much larger than the requests made.

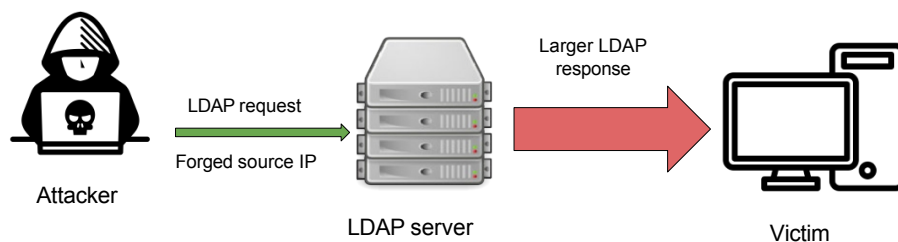


Figure 2.3: Schematic diagram of an LDAP reflection attack

2.2. Artificial intelligence

The science of artificial intelligence (AI) is devoted to the study and development of systems and programs capable of mimicking, simulating or sometimes even surpassing human intelligence in specific tasks. This is achieved through the use of complex algorithms and models that process large amounts of data and learn from them, allowing machines to perform tasks that would normally require human intervention.

Among the wide variety of tasks are automatic translation, object detection, speech recognition, autonomous driving, medical diagnosis, generation of new data, and many more.

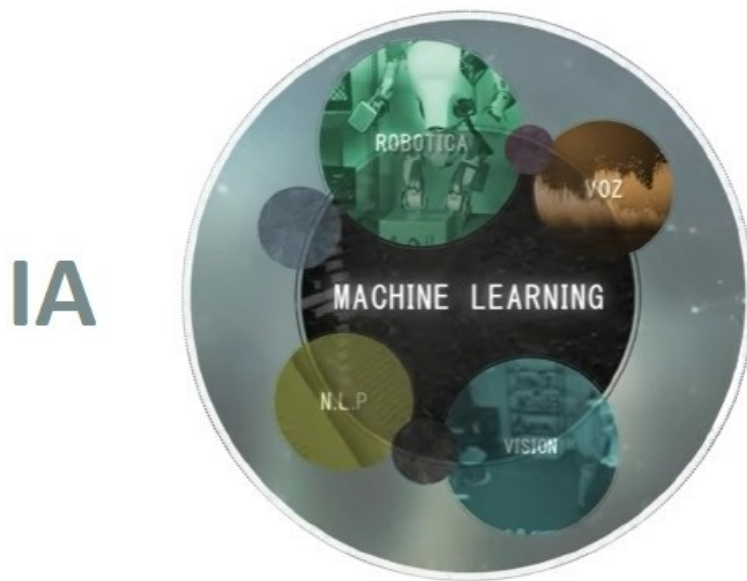


Figure 2.4: Conceptual map of artificial intelligence

Within artificial intelligence, there are a variety of techniques and approaches, including machine learning (also known as Machine Learning), natural language processing, fuzzy logic, computer vision, evolutionary algorithms, and expert systems. Each technique is adjusted based on the particular problem that needs to be solved and the data available. However, artificial intelligence also raises a number of problems. These include automated decision making by ethical agents. Some of these include automated decision-making by agents, such as the The latter has been the focus of a number of initiatives. The latter, ϕ , has been aggravated in particular by the advance of generative artificial intelligences, both in text and images, and in some cases even causing image rights problems.

2.2.1. Machine Learning

Machine Learning [5] is a branch of artificial intelligence whose main objective is to develop algorithms and techniques that enable artificial intelligence to

The machines learn from the data and make predictions without being explicitly programmed to do so. To achieve this, data sets are used that include input elements together with the corresponding desired outputs (labels or results). The central objective is to train a model with these data to generalise to other previously unseen data and to generate useful results.

The automatic learning procedure usually has several stages. First, a data preprocessing phase is carried out, which includes activities such as data cleaning, selection of relevant features and data normalisation. Subsequently, a suitable machine learning algorithm is chosen, which can be supervised, where the model is trained with input-output pairs, or unsupervised, where the model searches for patterns without labelled outputs. Once the algorithm has been chosen, it moves on to the training phase, where the model is provided with data to learn from and adjust to the patterns present. After training, the model is validated with the validation data set, a subset of the initial set that is split before training. With this data, the performance of the model is using different metrics. On. In case of not obtaining good results, the hyperparameters of the model can be adjusted and a new training can be done. When the model is ready to be used, it can be put into production to perform inference with new data.



Figure 2.5: Stages of automatic learning

2.2.1.1. Gradient boosting

Gradient boosting algorithms are a type of ensemble models, i.e. models that combine simple models to form a more robust and accurate model. Typically, the simple models used are decision trees [6] and this type of technique usually gives better results than random forest [7].

The underlying idea is in the iterative construction of decision trees, where each new tree is adjusted to correct the errors of the previous models. At each iteration, the algorithm adjusts a weak model to the errors of the previous models, i.e. to the difference between the actual values and the predictions of the previous models. In this way, the new weak model specialises in predicting the errors made by the previous models, which leads to a gradual improvement in the overall model.

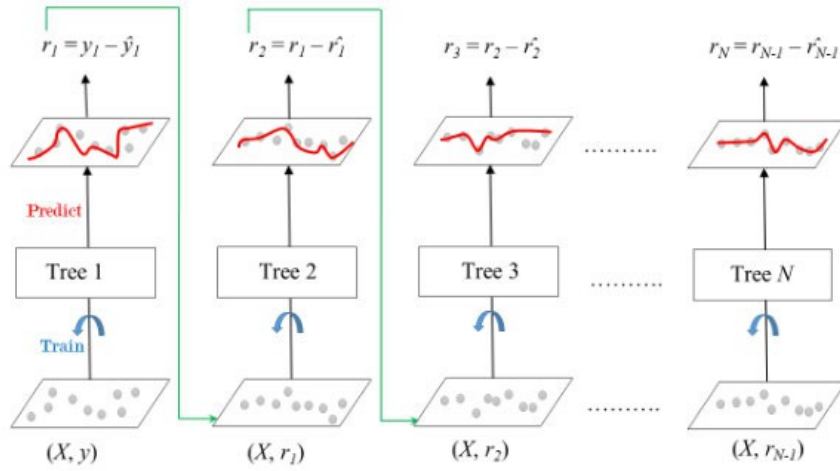


Figure 2.6: Basic operation of the gradient boosting algorithms

2.2.2. Deep Learning

Deep Learning [8] is a sub-area of Machine Learning that focuses on the development of learning algorithms based on artificial neural networks with multiple hidden layers. Unlike traditional Machine Learning models, Deep Learning models are able to learn relevant features directly from data in an automatic way. These models are capable of tackling much more complex tasks, such as natural language processing, speech and image recognition, autonomous driving, image and text generation, and automatic data mining.

These models are able to perform such tasks because of their ability to learn abstractly and to process large amounts of data. An example where this difference can be observed is a situation where a model has to recognise whether an image contains a car or not. For a traditional Machine Learning algorithm, it would first have to extract variables about the image, such as the number of wheels, the shape of the objects or their texture and colour. In contrast, a Deep Learning model would simply need the images, and it would be the model itself that would learn the most important features of the data.

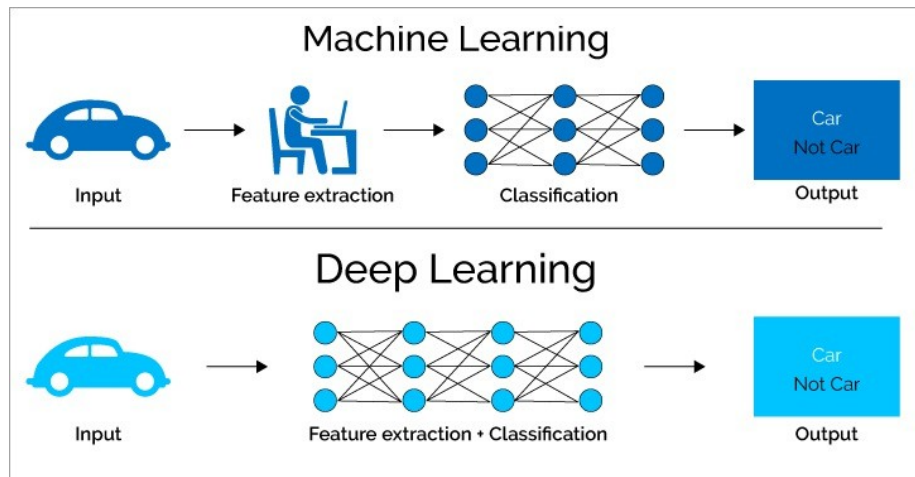


Figure 2.7: Comparison between Machine Learning and Deep Learning

Advances in computing power and the possibility of creating massive datasets have meant that very powerful models have been developed. These models have billions of parameters, such as Stable Diffusion [9] or LLMs [10] (Large Language Models) which have given rise to applications such as ChatGPT, and which are bringing about a great revolution for mankind.

2.2.2.1. Neural networks

Neural networks are a computational model based on the functioning of the brain. They are made up of neurons, which are organised in layers, and are connected to each other.

One of the most widely used architectures in neural networks is the multilayer perceptron, which consists of an input layer, one or more hidden layers and an output layer. Each neuron in a hidden layer is connected to all neurons in the previous layer and to all neurons in the next layer. In this way, the neural network receives the data through its input layer and, after performing its respective calculations, transmits it to the next layer. This process is repeated until the data reaches the output layer, the results of which are considered the output of the neural network. This process is known as forward propagation.

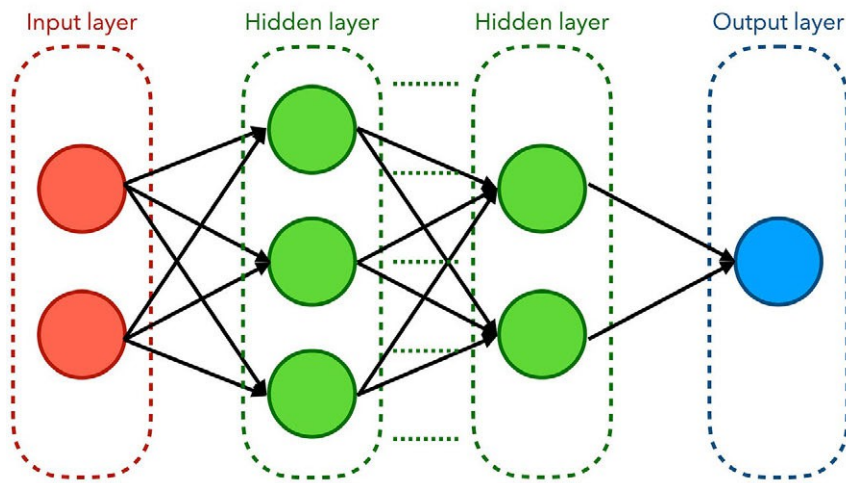


Figure 2.8: Image of a neural network

The calculations that each neuron performs are a linear combination of its coefficients, known as weights, and the data it receives. After this calculation, a non-linear function is applied, known as an activation function. The most common activation functions are the following:

Funci3n de Activaci3n	F3rmula	Range
Linear	$f(x) = x$	$(-\infty, \infty)$
Sigmoid	$f(x) = \frac{1}{1 + e^{-x}}$	$(0, 1)$
Hyperbolic tangent3lic	$f(x) = \tanh(x)$	$(-1, 1)$
ReLU (Rectified Linear Unit)	$f(x) = \max(0, x)$	$(0, \infty)$
Leaky ReLU	$f(x) = \max(0.01x, x)$	$(-\infty, \infty)$
Softmax	$f(x)_i = \frac{e^{x_i}}{\sum_j e^{x_j}}$	$(0, 1)$, sum to 1

Table 2.1: Main activation functions for neural networks

It should be noted that there are very diverse and complex neural network architectures. Some of them are convolutional neural networks (CNN) [11], recurrent neural networks (RNN) [12], generative adversarial neural networks (GAN) [13] and transformer networks [14]. Each of these architectures is designed to solve a specific task such as image analysis, sequence processing, synthetic data generation, and natural language processing.

2.3. Tools and libraries

The following tools have been used to carry out this work on:

- NumPy [15]: NumPy is a Python library that provides support for creating large multidimensional vectors and matrices, along with a large collection of high-level mathematical functions to operate efficiently with them.
- Pandas [16]: Pandas is a Python library specialised in data manipulation and analysis. It provides data structures and operations to manipulate numeric tables, it is like Python's Excel. It has been used to read CSV files from the dataset and modify the data to make it ready for input.
- Matplotlib [17]: Matplotlib is a library for generating two-dimensional graphs from data contained in lists or arrays in Python. It has been used to create the graphs on the training data of the neural network.
- Tensorflow [18] and Keras [19]: TensorFlow is an open-source platform for Machine Learning developed by Google. Keras runs on top of Tensorflow, and provides a high-level interface for creating neural networks.
- XGBoost [20]: XGBoost is an optimised library that implements gradient boosting algorithms, to be highly efficient, flexible and portable.
- Optuna [21]: Optuna is a framework for automatic optimisation of hyperparameters, specially for Machine Learning. It has been used with the gradient boosting model.
- HoloViews [22] and Hvplot [23]: HvPlot is a library built on top of HoloViews, and in the same way as matplotlib, they have been used for the creation of graphs [22].
- Vue.js [24]: Vue.js is a JavaScript framework for creating web interfaces with HTML, CSS and JavaScript. It provides a declarative and component-based programming model that facilitates the development of user interfaces. It has been used for the creation of the web interface in which the information of the training will be displayed.
- Flask [25]: Flask is a lightweight and flexible web framework for the development of web applications in Python, which facilitates the creation of websites and web services in a fast and easy way. With the help of this framework, the web server has been implemented.
- Pickle [26]: It is a library that allows to serialise and deserialise Python objects in a sequence of bytes, allowing to save these objects in files. It has been used to store the trained models, the label encoder and the standard scaler in files.

- Scikit-learn [27]: Scikit-learn is a Machine Learning library for Python that provides simple and efficient tools. Among all its functionalities, it has been used to calculate the metrics.
- Google Colab [28]: Google Colab is a product of Google Research. It allows any user to write and execute arbitrary Python code through the browser. It is especially suitable for Machine Learning and data analysis tasks. It has been used to process the data, train and analyse it. It has been used to process the data, train the models and create a notebook to make it easy for others to try this work.
- Ngrok [29]: Ngrok is a service that allows a local server to be created on a subdomain so that it can be viewed outside the LAN (Local Area Network), via the Internet. This has made it possible to access the server created on the machine running the Google Colab notebook.

2.4. Similar jobs

Below is a list of current work that has similar objectives to this one. On, a list of current work that has similar objectives to this one is shown:

- Enhanced Security Against Volumetric DDoS Attacks Using Adversarial Machine Learning [30]: This work addresses the challenge of detecting DDoS attacks by employing a generative adversarial network (GAN) to develop a robust attack detector. The proposed model can generate and classify benign and malicious synthetic traffic elements, which are very similar to the real ones.
- Detecting DDoS Attacks Using Machine Learning Techniques and Contemporary Intrusion Detection Dataset [31]: This paper presents a reliable dataset containing benign and common attack network flows, and explores the use of machine learning techniques to detect and track DDoS attacks.
- Machine Learning Techniques for Detecting DDOS Attacks [32]: This article discusses the use of machine learning techniques, specifically Random Forest, A decision trees on, SVM (Support vector machine), Naive Bayes and XGBoost, to detect DDoS attacks with a high level of accuracy on. It also uses the CICDDoS2019 dataset for analysis.
- Impact of Feature Selection Methods on Machine Learning-based for Detecting DDoS Attacks : Literature Review [33]: This study focuses on the impact of feature selection methods on machine learning-based DDoS attack detection [33]. It highlights the importance of feature engineering and its influence on the accuracy of machine learning solutions for DDoS attacks.

The novelty of this work, which does not include any of those mentioned in the list, is that it includes a tool for capturing traffic and a graphic interface where the operation of the system can be observed in real time.

Chapter 3

Project development

3.1. Dataset

In order to train a model capable of detecting attacks, a large amount of traffic data is necessary. For this purpose, the CICDDoS2019 dataset [34], has been chosen which contains benign traffic and common and up-to-date denial-of-service attacks, which resemble real world data (PCAPs). This dataset consists of Wireshark captures, however, it also includes CSV files containing results of an analysis of the traffic with flows tagged according to source and destination IPs, source and destination ports, protocols and attack type. This analysis has been obtained with CICFlowMeter [35], a traffic analysis tool capable of providing more than 80 variables. These CSV files will be used to train the models later and the description of each of the variables present can be found at <https://github.com/ahlashkari/CICFlowMeter/blob/master/ReadMe.txt>.

A notebook has been used to implement the data processing and model training. It can be accessed via https://colab.research.google.com/github/DavidRamosArchilla/Firewall-AI/blob/main/notebooks/Data_processing_and_model_training.ipynb.

3.1.1. Dataset processing

The size of the files that make up the dataset is quite large, in the order of GigaBytes, and this can be a limiting factor when loading them into memory to train the model. This is why it has been decided to carry out a preprocessing with the intention of reducing the size of the dataset. For this purpose, the code A.1 has been used

This preprocessing consists of the following: first, removing columns that would not be useful for training the model, such as source and destination IPs and source and destination ports, second, removing null (or nan) values and repeated elements in the dataset, then transforming the column data types into equivalent data types that take up less space (e.g. converting from float64 to float32) and finally saving the files to disk in Parquet format. Parquet is a columnar storage file format optimised for use with big data processing frameworks. It is able to provide efficient storage and processing of large data sets. Parquet files store data in a highly compressed and divisible form, which is ideal for reducing the size of the dataset on disk.

3.1.2. Analysis of the dataset

In order to get an overview of how the data is distributed within the dataset, a dimensional reduction algorithm has been applied to create a 2-dimensional image of the dataset. The algorithm used is t-sne [36] and the image obtained is as follows:

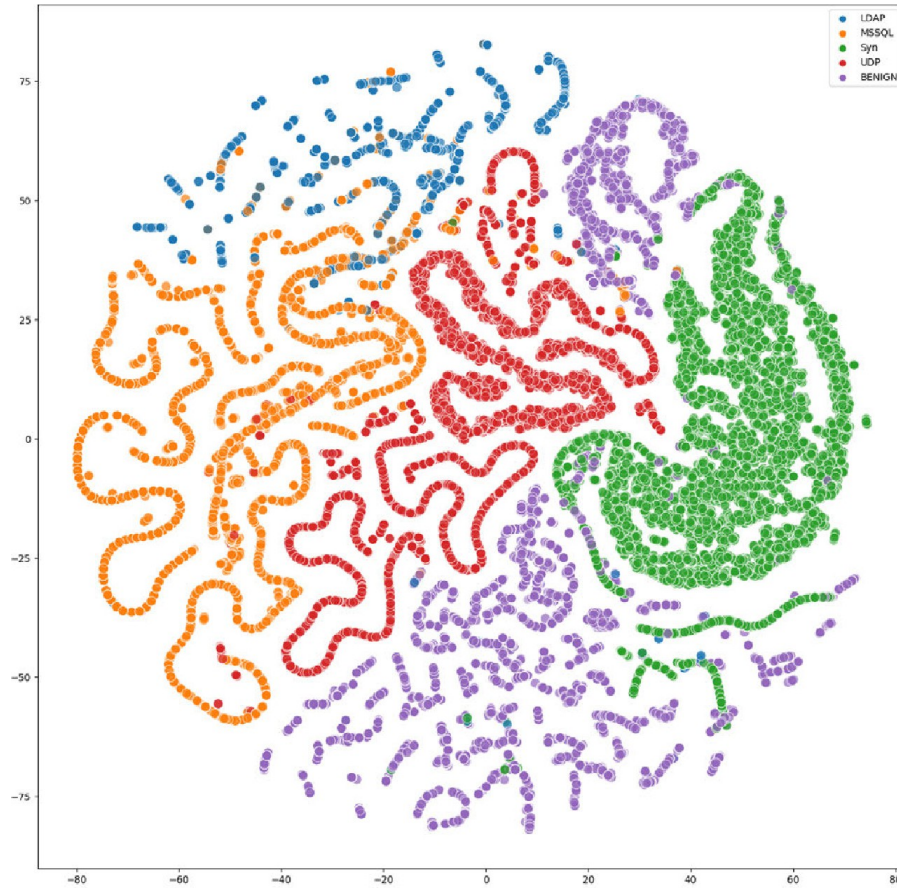


Figure 3.1: Visualisation of the dataset

On the other hand, the dataset contains a field called Label, which indicates the type of attack. The following diagram shows the amount of data for each type of attack:

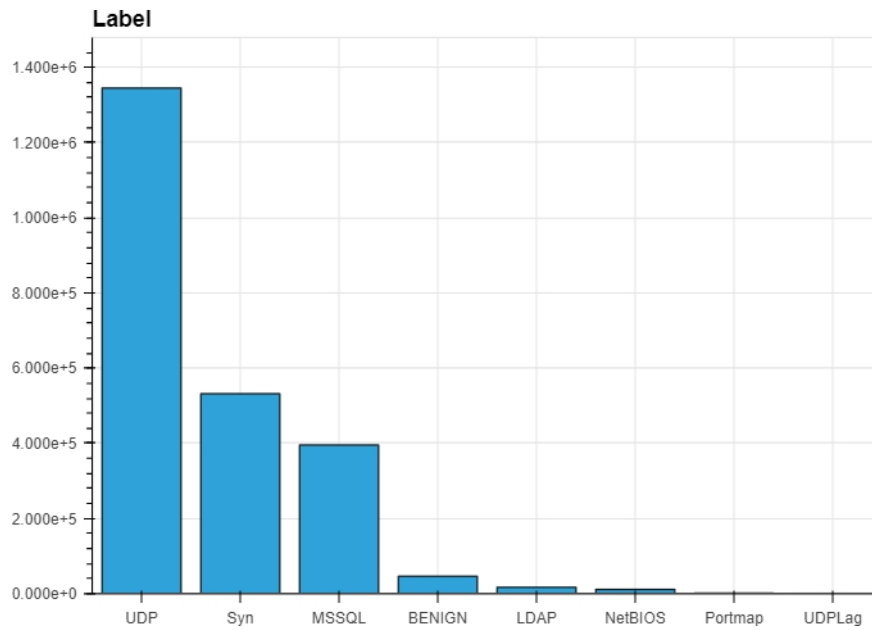


Figure 3.2: Distribution of the types of attacks on

As can be seen, the dataset is very unbalanced and there are classes that have too little data for a model to be able to learn about them. Therefore, the 3 classes with the fewest occurrences have been eliminated and the rest of the classes have been balanced, leaving the new dataset as follows:

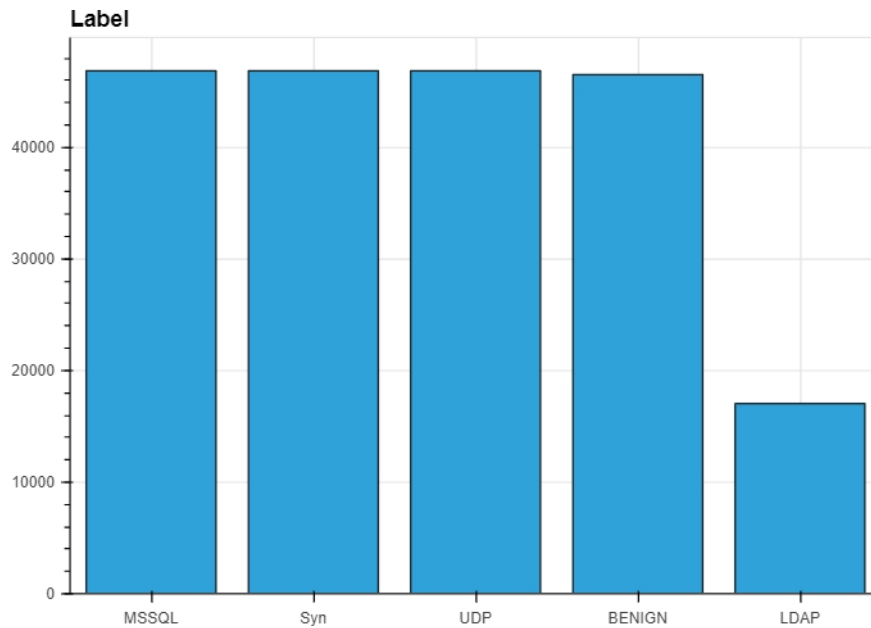


Figure 3.3: Distribution of the types of attacks after balancing on

3.2. Model training

Once the dataset has been processed, it will be almost ready for the training. One of the last remaining steps is the normalisation of the input variables. To normalise a variable, its mean is subtracted and divided by its standard deviation. Normalisation of the dataset is a common requirement for many machine learning estimators: they may misbehave if the individual characteristics do not look more or less like normally distributed standard data (e.g., Gaussian with mean 0 and unit variance).

For example, many elements used in the objective function of a learning algorithm assume that all features are centred around 0 and have a variance of the same order. If one feature has a variance that is several orders of magnitude larger than others, it may dominate the objective function and cause the estimator to fail to learn from other features correctly as expected, leading to a reduction in the accuracy of the model. To perform the normalisation, the `StandardScaler` class of the `scikit-learn` library has been used.

Finally, the labels indicating the type of attack come as text strings, so they have to be converted to numerical values so that the models can work with them. For this purpose, the `LabelEncoder` class of the `scikit-learn` library has been used.

Once these transformations have been carried out, it is possible to move on to the training phase of the models at .

3.2.1. Cross-validation

During the training of the models, the cross-validation technique is used, which consists of dividing the dataset into subsets called folds. Then, the model is trained and evaluated on different combinations of these folds to see how it behaves with each one of them.

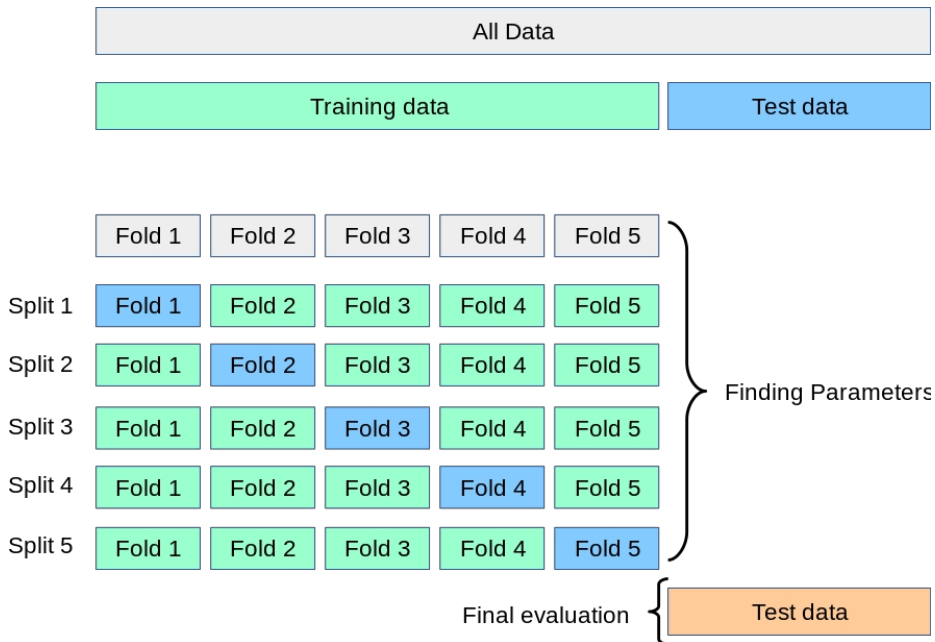


Figure 3.4: Cross-validation scheme

By evaluating the model on different subsets of the data, a more robust measure of the model's performance is obtained compared to the model's performance on a single training/testing division.

In addition, the possibility of a single training/testing division is reduced.

of bias due to a specific partitioning that accidentally favours the model.

3.2.2. Neural network

Specialised libraries have been used to build the neural network. Specifically, Keras, a high-level Deep Learning library written on top of TensorFlow, has been used. The type of neural network to be built is a multilayer perceptron.

The model consists of an input layer, which has a number equal to the number of input variables to receive to the neural network; two hidden layers, as fewer layers can lead to the model not learning enough and more layers increases the computational cost and can lead to overfitting; and an output layer with as many neurons as there are classes to predict, in this case, 5.

After several tests, by setting the number of neurons in each of the hidden layers to 64, good results have been obtained and the model is defined as follows:

Model: "model_1"

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	[(None, 75)]	0
dense_3 (Dense)	(None, 64)	4864
dense_4 (Dense)	(None, 64)	4160
dense_5 (Dense)	(None, 5)	325
Total params: 9349 (36.52 KB)		
Trainable params: 9349 (36.52 KB)		
Non-trainable params: 0 (0.00 Byte)		

As can be seen, the number of parameters of the neural network is 9349 and the it occupies is 36.52KB, so it will be quite efficient in making inferences even on CPUs.

As for the activation functions of the neural network layers, we have the following:

- ReLU: The activation function of ReLU (Rectified Linear Unit) is a non-linear function widely used in neural networks and deep learning models for hidden layers. The function of ReLU is defined mathematically as: $Relu(z) = \max(0, z)$.

This function of activation will have on the 2 hidden layers.

- Softmax: The activation function of softmax is commonly used in classification problems in machine learning and neural networks. Its main function is to convert a vector of real numbers into a vector of probabilities. It is particularly when dealing with multiclass classification problems, which makes it essential for this case. It is defined as follows:

$$Softmax(z_i) = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}$$

where K is the number of classes.

This is the activation function that will have to the output layer.

One thing to note, is the fact that the function of softmax returns a vector of probabilities. When the system is put into operation, this will be useful to know the confidence with which the model performs the predictions.

Because the dataset is not fully balanced (the LDAP class has fewer elements), we will set weights for each of the

classes in such a way that the model pays more attention to the classes with the highest weights. This measure should reduce the impact of the fact that the dataset is slightly unbalanced.

Finally, in order for the model error to be calculated correctly, a one-hot encoding must be applied to the target variable, in this case, the type of attack. This consists of converting the variable into a vector of the same size as the number of different elements in the variable, in this case 5. Vectors will have a 0 in all their elements except for the index of the class they represent.

The cost function used is the categorical crossentropy, and it is calculated using the `CategoricalCrossentropy` class of TensorFlow. Its mathematical formulation is as follows

$$L(y, \hat{y}) = - \sum_i y_i \log(\hat{y}_i)$$

Where:

- $L(y, \hat{y})$ represents the cross-entropy error (the name of the function).
- y represents the actual data to be predicted (labelled with one-hot encoding).
- \hat{y} represents the predictions.
- i iterates over all classes.

For training, the code of A.2 is used. It can be seen that cross-validation is applied with a total of 5 folds. Other important hyperparameters are the batch size (batch size), set to 32, and the learning rate, with a value of 0.0001. These values have been chosen to try to prevent the model from overfitting. The number of epochs has been set to 15, which is sufficient for the model to have a good accuracy.

In addition, at each iteration a graph is created with the errors in training and validation to see the evolution of the model, as well as a classification report, which contains metrics on the predictions of the validation set. For example, you can see the accuracy for each of the classes, which is a very good indicator to see how the model is working.

The following is the classification report and a graph showing the training and validation errors over the epochs for one of the folds. It is shown below:

Class	Precision	Recall	F1-Score	Support
BENIGN	0.99	1.00	1.00	7443
LDAP	0.84	0.97	0.90	2739
MSSQL	0.99	0.93	0.96	7495
Syn	1.00	1.00	1.00	7501
UDP	1.00	1.00	1.00	7487
Accuracy			0.98	32665
Macro Avg	0.96	0.98	0.97	32665
Weighted Avg	0.98	0.98	0.98	32665

Table 3.1: Neural network classification report on during cross-validation

Column Descriptions:

- **Class:** Represents the different types of attacks (or BENIGN if not).
- **Precision:** Ratio of true positives to the sum of true positives and false positives for each class. It is calculated with the formula:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

- **Recall:** Ratio of true positives to the sum of true positives and false negatives for each class. It is calculated with the formula:

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

- **F1-Score:** Harmonised mean of accuracy and recall for each class. It provides a balanced measure as it considers false and true positives. It is calculated with the formula:

$$F1\text{-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

- **Support:** Number of actual occurrences of each class.
- **Accuracy:** Overall correctness of the classification model. It is calculated with the formula:

$$\text{Accuracy} = \frac{\text{Correct Predictions}}{\text{Total Predictions}}$$

- **Macro Avg:** Average accuracy, recall and F1-Score for all classes.

- **Weighted Avg:** Weighted average of accuracy, recall and F1-Score considering the class imbalance.

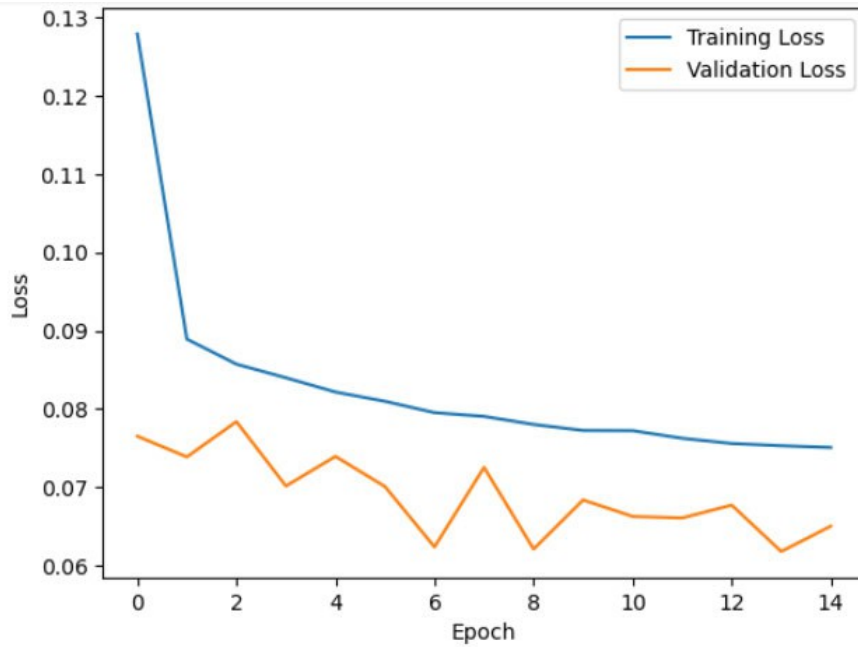


Figure 3.5: Error graph during cross-validation in the neural network After applying cross-validation, it can be observed that the model has an accuracy of 0.98 on The model is then trained with the entire training set. The results obtained are as follows:

Class	Precision	Recall	F1-Score	Support
BENIGN	1.00	0.99	0.99	9308
LDAP	0.82	0.96	0.89	3345
MSSQL	0.98	0.93	0.96	9388
Syn	1.00	1.00	1.00	9360
UDP	1.00	0.99	1.00	9431
Accuracy			0.98	40832
Macro Avg	0.96	0.98	0.97	40832
Weighted Avg	0.98	0.98	0.98	40832

Table 3.2: Classification report on final neural network

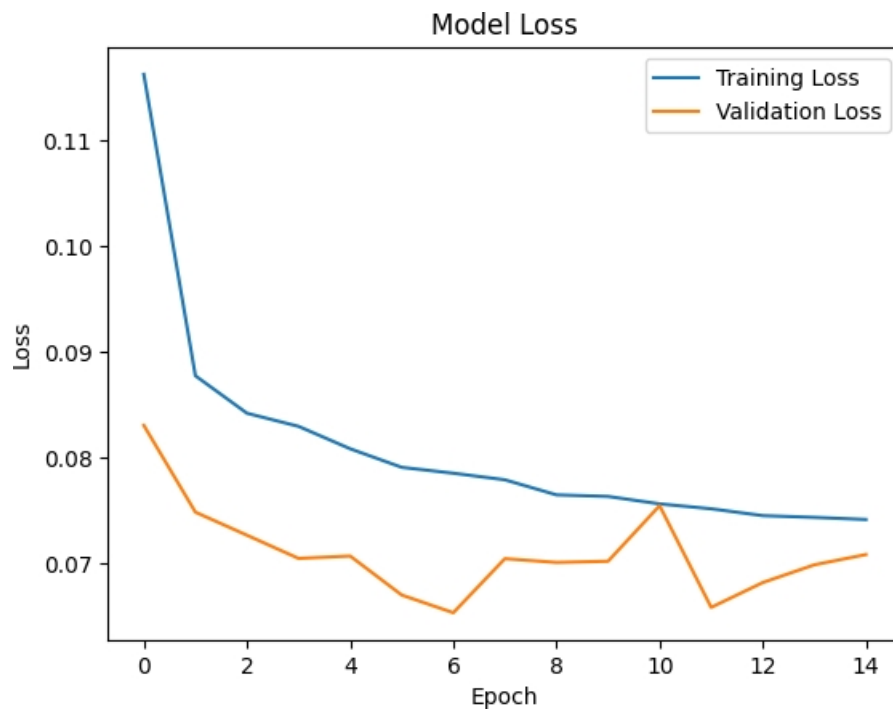


Figure 3.6: Error graph during final neural network training

Matriz de confusión

Real BENIGN	9232	53	0	23	0
Real LDAP	7	3216	102	20	0
Real MSSQL	6	627	8754	1	0
Real Syn	25	5	3	9324	3
Real UDP	0	1	53	1	9376
	Predicho BENIGN	Predicho LDAP	Predicho MSSQL	Predicho Syn	Predicho UDP

Figure 3.7: Confounding matrix 'on of the final neural network

This last graph is a confusion matrix^{on}. Each row indicates for each actual value, how many predictions the model has made for each of the classes, with the X-axis being the predictions and the Y-axis being the actual data. For example, the model predicted 102 times that a flow was an MSSQL attack when in fact it was LDAP, and predicted 627 times that a flow was an LDAP attack when in fact it was an MSSQL attack.

3.2.3. Gradient boosting model

The other model that has been trained is based on other Machine Learning techniques and it is a model based on gradient boosting algorithms [37]. Basically, this technique consists of a set of^{decision trees} that are trained in a sequential way to correct the errors of the previous model. Each^{tree} focuses on the remaining errors in the ensemble, and the final output is the weighted sum of the predictions of all^{trees}, gradually improving the accuracy^{of the model}. To create the model, the XGBClassifier class of the XGBoost library has been used.

In order to choose the hyperpar^{ameters}, the Optuna library has been used, a library created for this kind of purpose. It is simply a matter of defining an objective function to be optimised, providing search ranges for the hyperpar^{ameters} and choosing the number of iterations that the optimisation^{on} will take. During the search for hyperpar^{ameters}, it will be necessary to train the model, which will be done^{using cross-validation} as it was done with the neural network.

The corresponding code is A.3. With the trial object, a range is suggested from which to choose a value for the hyperpar^{ameter} in question[.]

After leaving the study running, the following hyperparameters are obtained[:]

```
{
  lambda': 3.242851557666658658e-
06, 'alpha':
1.8440002794817977e-06,
max_depth': 9,
eta': 0.1294208168667795,
'gamma': 8.19269827333313952e-
07,
'colsample_bytree': 0.14470448735955996,
min_child_weight': 6,
n_estimators': 1023
}
```

With the optuna.visualization module it is possible to obtain graphs about the optimisation^{on}. One of them is the following one, which shows the importance that each hyperpar^{ameter} has had during the optimisation^{on}:

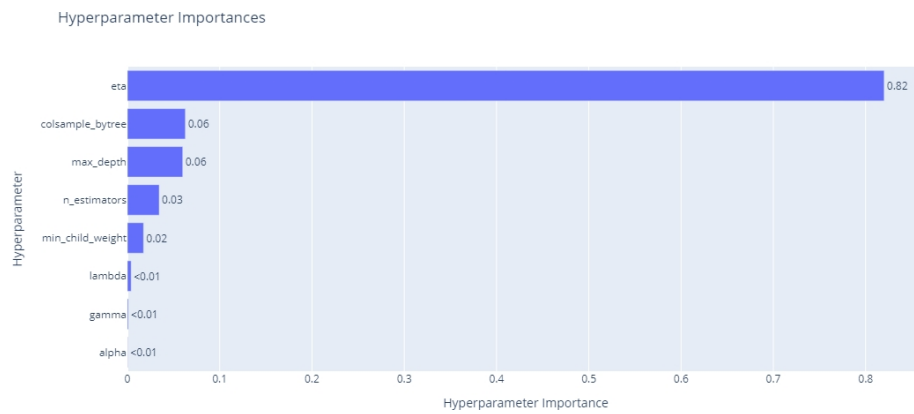


Figure 3.8: Importance of the hyperparameters

Other graphs can also be created which provide valuable information about the optimisation process as well:

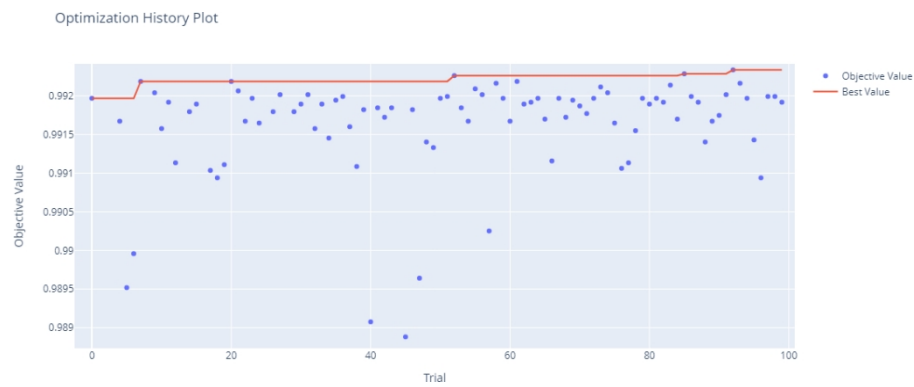


Figure 3.9: Historical record of the accuracy during optimisation

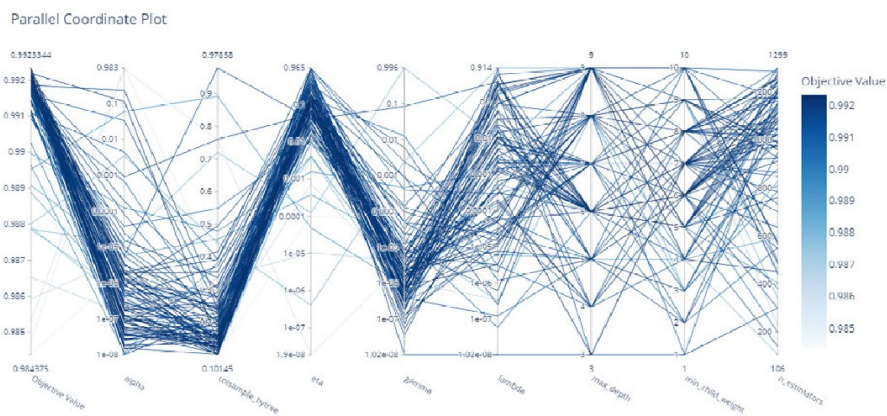


Figure 3.10: Distribution of the parameters tested during optimisation

The first one shows the evolution of the accuracy of the model during the experiment. The second one shows how the combinations of the different hyperparameters have been distributed and what accuracy has been achieved with them. This can be useful to modify the ranges over which the search of the hyperparameters is performed, since if the values are grouped a lot on one extreme it can indicate that the range has to be increased at that extreme, and the opposite happens if at one extreme there are few values chosen. For example, it can be seen that this occurs with the eta variable, which refers to the learning rate.

After obtaining some hyperparameters with which to start the model, the model is trained using the training set. The results after training are as follows:

Class	Precision	Recall	F1-Score	Support
BENIGN	1.00	1.00	1.00	9308
LDAP	0.95	0.97	0.96	3345
MSSQL	0.99	0.98	0.98	9388
Syn	1.00	1.00	1.00	9360
UDP	1.00	1.00	1.00	9431
Accuracy			0.99	40832
Macro Avg	0.99	0.99	0.99	40832
Weighted Avg	0.99	0.99	0.99	40832

Table 3.3: Classification report on the gradient boosting model

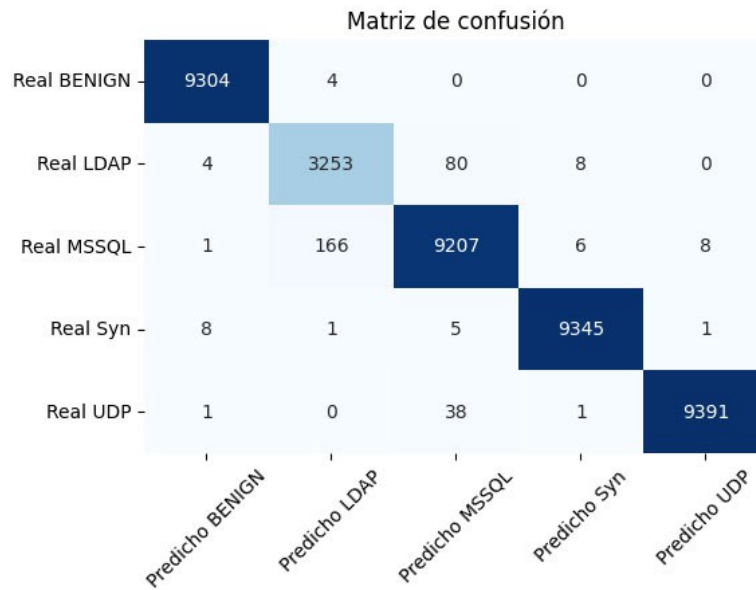


Figure 3.11: Confusion matrix of the gradient boosting model

One of the positive aspects of this model is that, as it is based on decision trees, it is possible to see on which variables it relies to make predictions. This is why it is possible to see how important each of the variables in the dataset is, something that can be very useful when selecting the input variables for a model, if necessary. Below is a graph showing the 25 most important variables for this model.

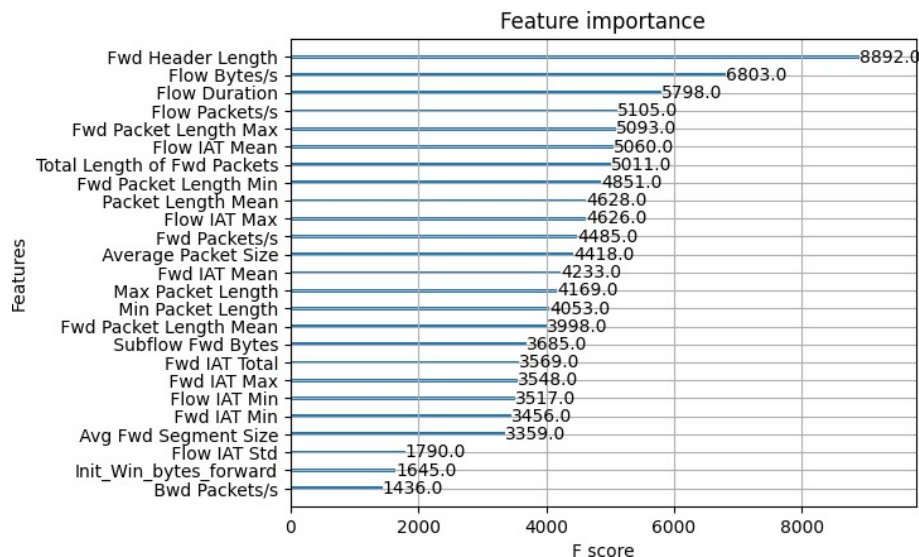


Figure 3.12: Importance of input variables

3.3. Traffic Capture Tool *fico*

In order to run the trained models, it is necessary to capture the network traffic in some way so that it can be analysed. The same institution that created the dataset used in this work, the Canadian Institute for Cybersecurity (CiC), also has a traffic capture tool, which they have called *CicFlowmeter*, already mentioned above. This tool, in addition to having the ability to produce CSV files containing traffic variables from pcap files (a common format for storing packet captures) or similar, can also read packets received by a network interface in real time. This makes it the ideal tool for the system to be developed, as it will allow to capture the traffic and, in turn, calculate the input variables for the model, all in real time.

However, there is a drawback, the tool is written in Java and complicates the integration with the rest of the system. Fortunately, the community has created a similar version for Python [38]. As a starting point it is fine, but it is not sufficient for what we are looking for, so, from a fork, a Python library has been created to satisfy the needs of this work.

3.3.1. *PyFlowmeter*

PyFlowmeter [39] is the name of the library that has been created. To install it, it can be done in a very simple way by executing: `pip install pyflowmeter`. As already mentioned, modifications have been made to the original project to adapt it to this work. Some of its functionalities are:

- Real-time packet capture. This tool will be able to capture packets in real time by simply specifying one or more interfaces to listen on. To obtain only the necessary packets, a filter, which is "ip and (tcp or udp)", has been added to it.
- Simulation of network traffic from pcap files. Using files containing network traffic data, the tool can simulate the network traffic, as if it were being received. This is very useful when testing on the system.
- Sending captured packets to a server. Periodically, the tool may send the traffic data it has stored to a server to be specified. The sending of this data is done in the background and can be either real time captured traffic through the network interfaces or simulated traffic.
- Generation of CSV files on the network traffic. The tool is able to generate a CSV file with more than 70 variables on the network traffic, either in real time or simulated from a file. In the case of a simulation file, this functionality can be seen as the conversion of traffic packet files to CSV files with variables on the traffic.

It should be noted that the data PyFlowmeter collects is stored in the form of flows. Network flows are data streams that represent the communication between devices in a network. Each flow is composed of information such as source and destination IP addresses, ports, protocols and the amount of data transmitted. In this way, each line of the CSV files corresponds to a flow and the data to be sent to the server will be a list of flows.

The following code snippets show examples of how this library can be used:

- Get CSV analysis of a pcap file: A.4
- Scan packets in real time from the network interface and send the data to a server every 5 seconds: A.5
- Simulate traffic from a pcap file: A.6

All of this and more can be seen in detail at <https://pypi.org/project/pyflowmeter/>

3.4. System architecture

The system is made up of different components or sub-systems as shown in the following diagram:

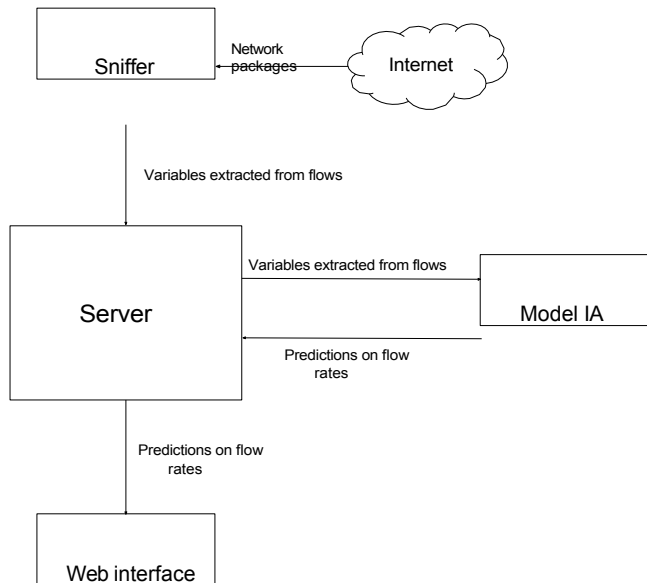


Figure 3.13: Diagram on the system

First, the sniffer (or packet analyser) will capture all the packets that are received by the selected network interfaces and send to the server the variables about the network flow data. The server will then make use of artificial intelligence models to analyse the flows and determine whether they are an attack or not. This is done through the model component, which will be integrated in the server. The server will display, through an http service, a web application in which you can see the traffic received and the predictions made by the model on it, among other things.

The system has been developed on WSL (Windows Subsystem for Linux), with a Debian distribution (on Debian). All the code is available on github (<https://github.com/DavidRamosArchilla/Firewall-AI>) so that it can be easily replicated.

3.4.1. Server

The server is the main component of the system and it will be in charge of sending and receiving data. The Python Flask library has been used for its implementation.

For simplicity, the sniffer has been included in the server. The alternative to this would be to create a machine that would run the sniffer and communicate with another machine that would be running the server. The server would send the predictions back to the sniffer and the sniffer would act as a firewall. This sniffer is created using the previously explained tool, PyFlowmeter. The way it works is as follows: the sniffer captures all the packets it receives on the specified interface and groups them into flows. For each of these flows, the sniffer is able to obtain variables about them, which are the same variables that were used for training. The sniffer has to send this data to the server. For this purpose, there is a thread that sends data about the flows to the server every few seconds. The frequency with which the data is sent can be configured, and by default it is sent every second.

As it is included in the server, the sniffer can be started from the web interface. For this purpose an endpoint has been created, `/start sniffer`, which could be used to start the sniffer to read the network traffic in real time or to simulate traffic from files of your choice (it will be a useful for testing purposes).

In addition, the server will have other endpoints such as `/send traffic` and `/get data`. The endpoint `/send traffic` will be a POST type, and that is where the server will receive the streams obtained by the sniffer. After receiving the flows, the first thing the server does is to convert the data to a Python dictionary. It then passes this data to the model for it to process and make predictions. After obtaining these predictions, the server stores them in a list. On the other hand, `/get data`, which is of type GET, returns to a list with all the flows and the prediction made by the artificial intelligence model, which will be important when building the web interface.

Finally, to start the server, the command `'sudo venv/bin/python app.py'` must be executed. About this command, it should be noted that it is necessary to have superuser permissions for the sniffer to be able to capture network packets and that the Python path indicated is from a virtual environment. A Python virtual environment is an isolated directory that allows the installation of the dependencies of a project, without needing to install them in the whole system.

To replicate the server, the following steps can be followed: First, you have to install the dependencies and clone the project repository:

```
sudo apt install libpcap0.8
sudo apt install tcpdump
git clone https://github.com/DavidRamosArchilla/Firewall-AI.git
cd Firewall-AI
```

Next, it is advisable to create a virtual Python environment

```
python -m venv venv
source venv/bin/activate
```

and install the libraries:

```
pip install -q --upgrade pip
pip install -q
pyflowmeter
pip install -q -r Firewall-AI/requirements.txt
```

Finally, the above-mentioned command is executed :

```
sudo venv/bin/python app.py
```

3.4.2. Artificial intelligence model

The module of the artificial intelligence model is in charge of making predictions about whether traffic flows are attacks or not. To do this, the input data must first be normalised using the same "normaliser" as during training. Then, inference is performed with the model, which produces a vector of integers between 0 and the number of classes minus 1 (4 in this case) with as many elements as flows are passed as input. These integers, which represent the class of the flow, are converted to their corresponding labels using the same LabelEncoder object that was used during training.

All this is encapsulated in a class, and it is the server that creates an object of this class to make the predictions.

The corresponding code is A.10

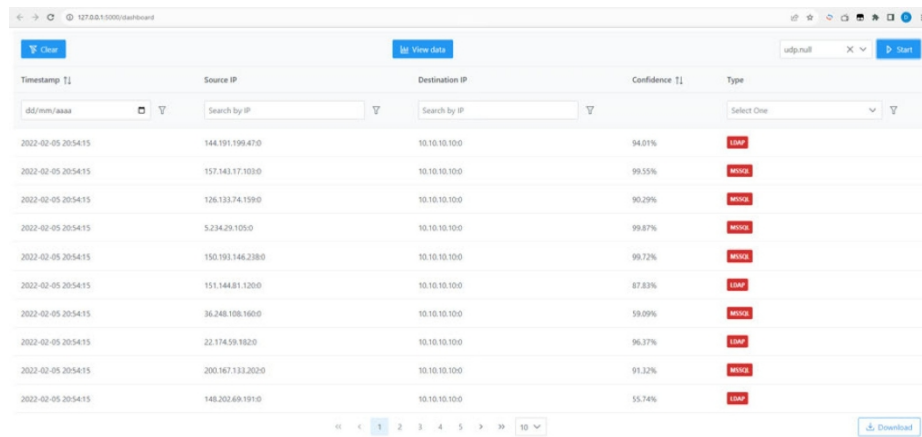
3.4.3. Web interface

In the web interface you can see information about the network traffic and what the model predicts about it. For its implementation, we have used Vue.js, a JavaScript framework for the creation of graphical interfaces. We have chosen to make a single page application (SPA), which is a type of web applications that loads a initial page from the server and then dynamically updates its content. For the creation of some of the components, use has been made of PrimeVue, a UI component library for Vue.js, which provides a variety of predefined components and ready-to-use styles.

The interface has 2 views, which would be controlled by an A.9 , router namely dashboard and traffic-analysis. Both views obtain the necessary data through a request to the get data endpoint of the server, already explained above.

The first one is a table showing the network traffic. The table includes the source and destination IP addresses, as well as the ports, the date or timestamp, the confidence with which the model has made the prediction on and the type of flow, which will appear in one colour or another depending on whether it is an attack or not. The table can be sorted by timestamp or by the confidence of the prediction on, and the elements can also be filtered by IP addresses and ports, the timestamp and the type of flow.

On the other hand, there are several buttons: at the bottom right there is a button on that allows downloading the table in CSV format, at the top left there is a button on that removes the filters that have been set, at the top right there is a drop-down and a button on that allow starting the sniffer for real-time traffic or with preset pcap files, and at the top centre there is a button on to open the other view. Below on is a screenshot of the dashboard view:



The screenshot shows a web interface for a network traffic dashboard. At the top, there are buttons for 'Clear', 'View data', and a 'Start' button next to a 'udpnull' dropdown. Below these are filters for 'Timestamp', 'Source IP', 'Destination IP', 'Confidence', and 'Type'. The main table displays 10 rows of traffic data. At the bottom right, there is a 'Download' button.

Timestamp	Source IP	Destination IP	Confidence	Type
2022-02-05 20:54:15	144.191.199.470	10.10.10.100	94.01%	LOAF
2022-02-05 20:54:15	157.143.17.1030	10.10.10.100	99.55%	WIDS
2022-02-05 20:54:15	126.133.74.1190	10.10.10.100	90.29%	WIDS
2022-02-05 20:54:15	5.234.29.1050	10.10.10.100	99.67%	WIDS
2022-02-05 20:54:15	150.193.146.2380	10.10.10.100	99.72%	WIDS
2022-02-05 20:54:15	151.144.81.3200	10.10.10.100	87.83%	LOAF
2022-02-05 20:54:15	36.248.106.1600	10.10.10.100	59.09%	WIDS
2022-02-05 20:54:15	22.174.59.1820	10.10.10.100	96.37%	LOAF
2022-02-05 20:54:15	200.167.133.2020	10.10.10.100	91.32%	WIDS
2022-02-05 20:54:15	148.202.69.1910	10.10.10.100	55.74%	LOAF

Figure 3.14: Dashboard view

The other view is traffic-analysis. This view contains 2 charts that show the information that appears in the table of the dashboard view, but in a more visual form. The Chart.js library is used to create the charts, which is integrated with PrimeVue. The graph on the left shows the amount of each type of flow that has been detected and the graph on the right shows the types of flow that have been detected over time. A screen shot of this view is shown below:

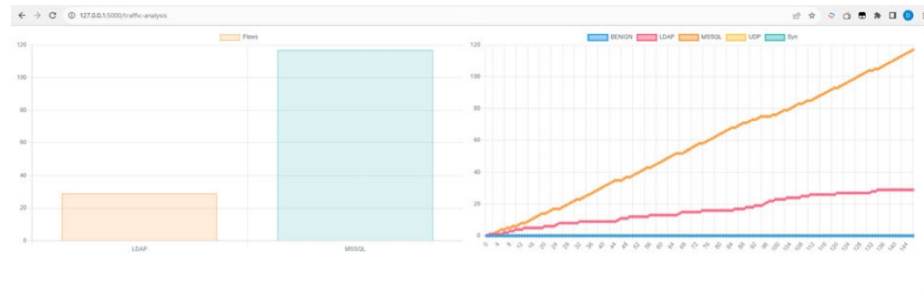


Figure 3.15: Traffic-analysis view

Chapter 4

Results

After finishing the implementation of the system, it is time to test that the system works correctly. So that it can be tested without the need to install the server and its dependencies in a machine, a notebook has been created in Google Colab that deploys the server. In this way, the server runs in the machine provided by Google Colab, and with the help of Ngrok, the web interface can be accessed via a URL. Ngrok is a service that creates a secure tunnel to a server running locally and provides a public access URL that allows external users to access its server. Without this feature, it would not be possible to access the web interface of the server running on the notebook. The link to access such a notebook is https://colab.research.google.com/github/DavidRamosArchilla/Firewall-AI/blob/main/notebooks/server_firewall_ai.ipynb

To use Ngrok services it is necessary to have an authentication token. It can be obtained free of charge after creating an account on Ngrok at the following link <https://dashboard.ngrok.com/get-started/your-auth-token>.

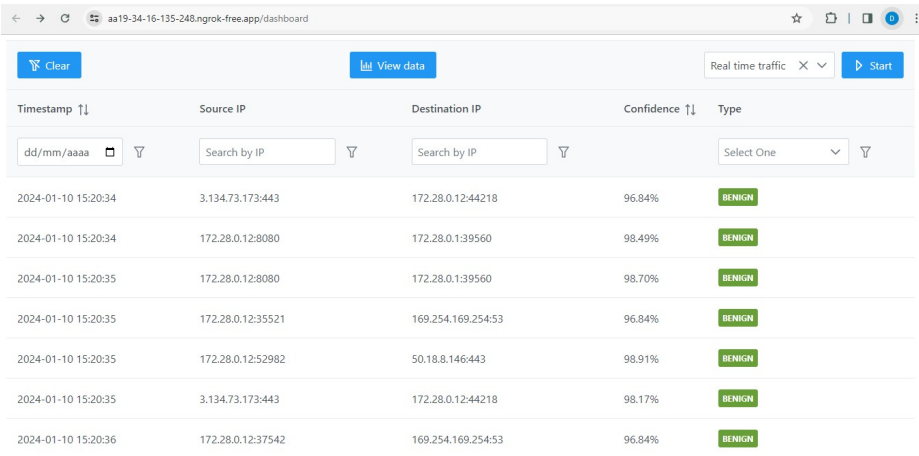
4.1. System in operation

To evaluate the performance of the system, attacks will be simulated using the same tool that is used to capture the traffic, Pyflowmeter. This functionality is very interesting, since from files that contain data about the packets of a network, the same traffic can be simulated. The files with which the experiments will be carried out come from the StopDDoS project (<https://github.com/StopDDoS/packet-captures>).

On the other hand, for the execution of the system, the model will be used trained with XGBoost, since, as we have seen in previous sections, it obtains better metrics for the validation set than the neural network. In order to store the trained model in a file and subsequently upload it to the server and make it ready to perform inferences, Pickle, a library for serialising and deserialising Python objects, has been used.

4.1.1. Traffic in real time

For the first experiment carried out, the system was run to analyse the traffic in real time. For this situation, it is useful to create the server with the Google Colab notebook, since the machine where it is running is constantly receiving and sending packets (although Google actually uses containers). After leaving the server running for a period of time, the following results are obtained:



Timestamp ↑↓	Source IP	Destination IP	Confidence ↑↓	Type
dd/mm/aaaa	Search by IP	Search by IP		Select One
2024-01-10 15:20:34	3.134.73.173:443	172.28.0.12:44218	96.84%	BENIGN
2024-01-10 15:20:34	172.28.0.12:8080	172.28.0.1:39560	98.49%	BENIGN
2024-01-10 15:20:35	172.28.0.12:8080	172.28.0.1:39560	98.70%	BENIGN
2024-01-10 15:20:35	172.28.0.12:35521	169.254.169.254:53	96.84%	BENIGN
2024-01-10 15:20:35	172.28.0.12:52982	50.18.8.146:443	98.91%	BENIGN
2024-01-10 15:20:35	3.134.73.173:443	172.28.0.12:44218	98.17%	BENIGN
2024-01-10 15:20:36	172.28.0.12:37542	169.254.169.254:53	96.84%	BENIGN

Figure 4.1: Table of the analysis of the received real-time traffic

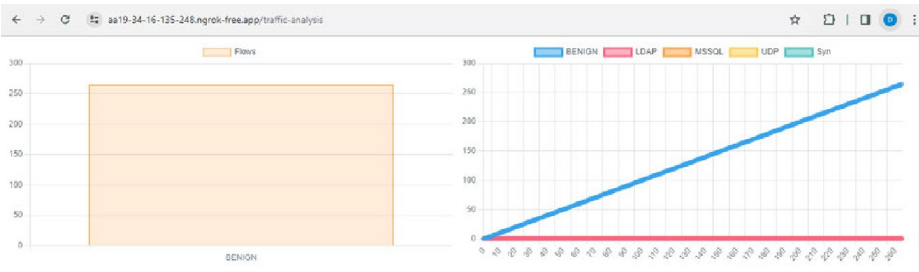


Figure 4.2: Graphs of the analysis of the real time traffic received in real time As can be seen, more than 200 network flows have been received and the system has predicted for all of them that they were benign, i.e. that they were not attacks. This indicates that the system is making the predictions correctly.

4.1.2. LDAP attack

As a first simulated attack, an attack with LDAP on reflexi on has been chosen. The following on shows the results that the system offers after the analysis. alisis:

<div>Clear</div> <div>View data</div> <div>UDP LDAP</div> <div>Start</div>				
Timestamp \updownarrow	Source IP	Destination IP	Confidence \updownarrow	Type
<div>dd/mm/aaaa</div>	<div>Search by IP</div>	<div>Search by IP</div>	<div>Select One</div>	
2021-05-13 19:53:35	180.176.178.132:123	10.10.10.10:23156	83.76%	MSSQL
2021-05-13 19:53:35	95.154.71.36:53	10.10.10.10:23156	65.99%	LDAP
2021-05-13 19:53:35	207.154.214.216:53	10.10.10.10:23156	87.83%	LDAP
2021-05-13 19:53:35	204.99.128.105:11211	10.10.10.10:23156	98.24%	LDAP
2021-05-13 19:53:35	182.50.64.181:53	10.10.10.10:23156	86.03%	MSSQL
2021-05-13 19:53:35	171.231.11.9:123	10.10.10.10:23156	99.93%	UDP
2021-05-13 19:53:35	61.50.187.62:11211	10.10.10.10:23156	79.82%	LDAP

Figure 4.3: Table of the analysis of an LDAP attack

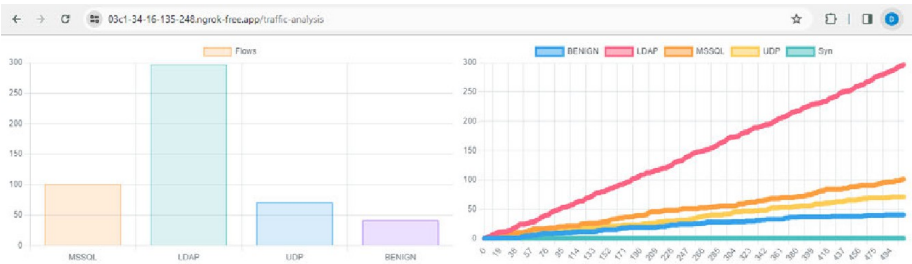
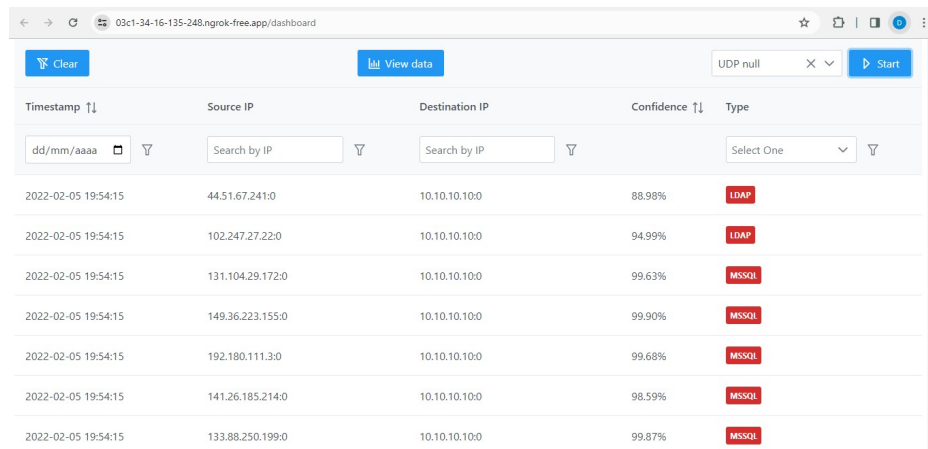


Figure 4.4: Graphs of the analysis of an LDAP attack

After running the simulation, the system detects that the vast majority of the flows are LDAP type attacks, which is correct. It also detects a smaller part of the flows as MS SQL and UDP attacks. This is due to the similarity between these types of attacks. Finally, there are some flows that are detected as benign, but this is not a problem as it is clear from the total number of flows received that the system is able to detect the attack.

4.1.3. UDP null attack

Another attack that has been simulated is a UDP null attack. UDP null is a type of attack based on sending empty UDP packets to a server or network device, with the aim of overloading it and making it stop working. These are the results after the simulation of this attack:



Timestamp	Source IP	Destination IP	Confidence	Type
2022-02-05 19:54:15	44.51.67.241:0	10.10.10.10:0	88.98%	LDAP
2022-02-05 19:54:15	102.247.27.22:0	10.10.10.10:0	94.99%	LDAP
2022-02-05 19:54:15	131.104.29.172:0	10.10.10.10:0	99.63%	MSSQL
2022-02-05 19:54:15	149.36.223.155:0	10.10.10.10:0	99.90%	MSSQL
2022-02-05 19:54:15	192.180.111.3:0	10.10.10.10:0	99.68%	MSSQL
2022-02-05 19:54:15	141.26.185.214:0	10.10.10.10:0	98.59%	MSSQL
2022-02-05 19:54:15	133.88.250.199:0	10.10.10.10:0	99.87%	MSSQL

Figure 4.5: Analysis table of a UDP null attack

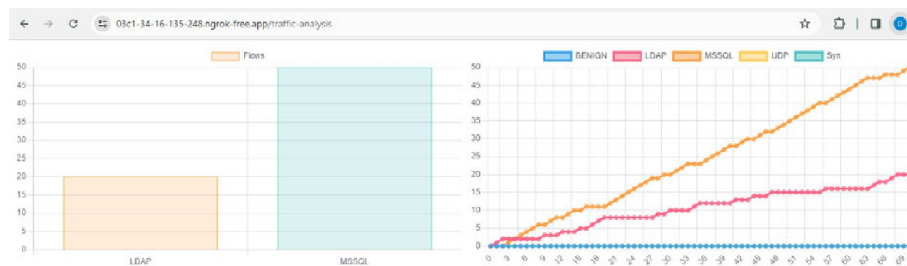
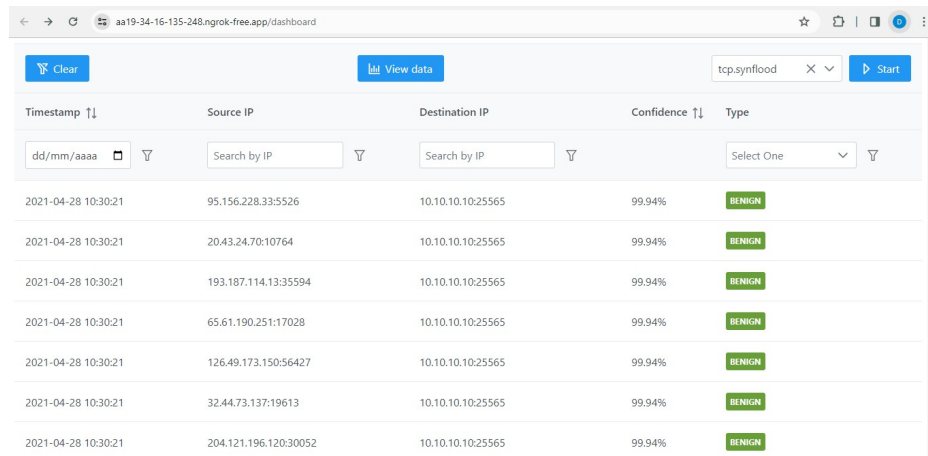


Figure 4.6: Graphs of the analysis of a UDP null attack

In this attack, it can be seen that the system detected that all the flows corresponded to an attack. However, the predictions it made were that the flows were LDAP and MS SQL attacks. This may be due to the fact that the simulated attack type does not sufficiently resemble any of the types in the training dataset and therefore the model is behaving in this way. In any case, the system has been able to detect that the traffic received was an attack.

4.1.4. TCP SYN flood attack

Although the system has been able to detect the above attacks, there are some attacks that escape detection because they are difficult to detect. A good example is the TCP SYN flood attacks, since the packets sent in these attacks appear to be legitimate. The results obtained after performing this attack are shown below:



Timestamp ↑↓	Source IP	Destination IP	Confidence ↑↓	Type
2021-04-28 10:30:21	95.156.228.33:5526	10.10.10.10:25565	99.94%	BENIGN
2021-04-28 10:30:21	20.43.24.70:10764	10.10.10.10:25565	99.94%	BENIGN
2021-04-28 10:30:21	193.187.114.13:35594	10.10.10.10:25565	99.94%	BENIGN
2021-04-28 10:30:21	65.61.190.251:17028	10.10.10.10:25565	99.94%	BENIGN
2021-04-28 10:30:21	126.49.173.150:56427	10.10.10.10:25565	99.94%	BENIGN
2021-04-28 10:30:21	32.44.73.137:19613	10.10.10.10:25565	99.94%	BENIGN
2021-04-28 10:30:21	204.121.196.120:30052	10.10.10.10:25565	99.94%	BENIGN

Figure 4.7: Table of the analysis of a TCP SYN flood attack

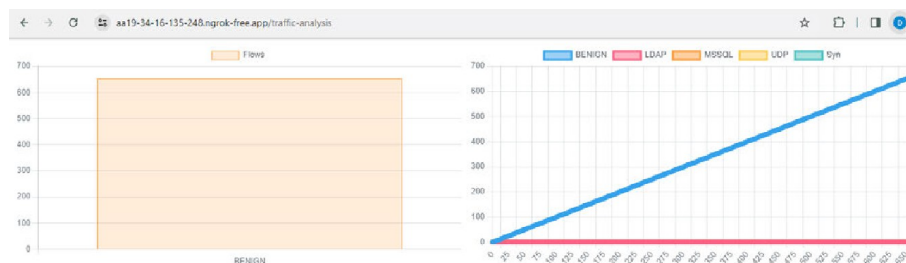


Figure 4.8: Graphs of the analysis of a TCP SYN flood attack

As can be seen, the system is not able to detect the attack and detects the network flows as benign.

Chapter 5

Conclusions and future work

5.1. Conclusions

This work proposes a solution to have a system that can analyse the network traffic in real time, in order to detect denial of service attacks. To achieve this, an artificial intelligence model is in charge of making predictions about whether or not an attack is taking place. To create the model, a series of experiments have been carried out to achieve the best possible version, obtaining more than satisfactory metrics, such as an accuracy of 99%.

After testing the system by running simulations of different types of attacks, it is found that the system is able to detect them in most cases. However, there are some situations where the attack traffic resembles legitimate traffic in such a way that it appears indistinguishable, and the system is not able to detect the attack. This indicates that, despite meeting the objectives, there is room for improvement and other techniques may need to be employed.

Finally, it should be noted that the dataset that has been used to train the models is not perfect. This means that it has been artificially generated for this purpose, and during its creation, bias may have been introduced during its creation. Avoiding this is a very complex task, and this may be one explanation why the system does not detect some attacks even though it has such a high accuracy on the validation dataset.

5.2. Social and environmental impact

The environmental impact of this work could be considered as null, as it is not related to this issue. Regarding the social impact, the fact that the number of DDoS attacks can be reduced is positive, as they are considered as malicious attacks. At the same time, the perpetrators of these attacks would no longer benefit from it.

On the other hand, this work could be aligned with the UN Sustainable Development Goals (SDGs). For a start, the use of advanced technologies such as machine learning and artificial intelligence to detect cyber-attacks aligns with SDG 9 (Industry, innovation and infrastructure), as it fosters innovation in technological infrastructures and promotes the development of secure information systems. In relation to this, cyber security is fundamental for stability and peace in cyberspace. As this work deals with the detection of DDoS attacks, it contributes to promoting peace in the digital environment, which relates to SDG 16 (Peace, justice and institutions).

sólidas).

5.3. Future lines

Although the objectives set out for this work have been met, some possible future lines of research are presented below. On:

- Increase the number of attack types to be detected to create a more complete system. Although other types of attacks are included in the dataset, the amount of data on them is not sufficient to train a model.
- Use tools that simulate real attacks to check the correct functioning of the system.
- Improve the look and feel of the web application and add additional pages, such as a home page.
- Create a user system to access the data on the web page. In this way only the users registered in the system would have access to them. This has not been carried out since in the proposed architecture the server is located within the local network, and therefore it would be easier to control who can access it.
- Modify the architecture so that the sniffer (packet analyser) and the server run on different machines. In this way, the machine running the sniffer would act as a firewall and the server would simply perform the calculations to predict whether or not the traffic being analysed is an attack or not.

Bibliography

- [1] O. Yoachimik and J. Pacheco, "Ddos threat report 2023 q1 (en)," Tech. Rep., March 2023. [Online]. Available: <https://blog.cloudflare.com/es-en/ddos-threat-report-2023-q1-en-es-en/>
- [2] C. Systems, "cisco Annual Report 2018," Tech. Rep, February 2018. [Online]. Available: https://www.cisco.com/c/dam/global/es_mx/solutions/pdf/report-annual-cisco-2018-espan.pdf
- [3] A. W. Services, "Threat landscape report - q1 2020," Tech. Rep., May 2020. [Online]. Available: <https://aws-shield-tlr.s3.amazonaws.com/2020-Q1AWS.Shield.TLR.pdf>
- [4] G. Cloud, "How google cloud blocked the largest layer 7 ddos attack at 46 million rps," Tech. Rep, June 2022. [Online]. Available: <https://cloud.google.com/blog/products/identity-security/how-google-cloud-blocked-largest-layer-7-ddos-attack-at-46-million-rps>
- [5] Z.-H. Zhou, *Machine learning*. Springer Nature, 2021.
- [6] C. Kingsford and S. L. Salzberg, "What are decision trees?" *Nature biotechnology*, vol. 26, no. 9, pp. 1011-1013, 2008.
- [7] L. Breiman, "Random forests," *Machine learning*, vol. 45, pp. 5-32, 2001.
- [8] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436-444, 2015.
- [9] R. R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 10 684-10 695.
- [10] Wikipedia contributors, "Llm (large language model)," 2023, [Online; accessed 2-June-2023]. [Online]. Available: [https://es.wikipedia.org/wiki/LLM_\(large_language_model\)](https://es.wikipedia.org/wiki/LLM_(large_language_model)).
- [11] K. O'Shea and R. Nash, "An introduction to convolutional neural networks," *arXiv preprint arXiv:1511.08458*, 2015.
- [12] A. Sherstinsky, "Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network," *Physica D: Nonlinear Phenomena*, vol. 404, p. 132306, 2020.
- [13] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *Communications of the ACM*, vol. 63, no. 11, pp. 139-144, 2020.
- [14] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

- [15] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. Fernández del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, p. 357-362, 2020.
- [16] W. McKinney *et al.*, "Data structures for statistical computing in python," in *Proceedings of the 9th Python in Science Conference*, vol. 445. Austin, TX, 2010, pp. 51-56.
- [17] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in science & engineering*, vol. 9, no. 3, pp. 90-95, 2007.
- [18] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: <https://www.tensorflow.org/>
- [19] F. Chollet *et al.*, "Keras," <https://keras.io>, 2015.
- [20] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16. ACM, Aug. 2016. [Online]. Available: <http://dx.doi.org/10.1145/2939672.2939785>
- [21] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," 2019.
- [22] HoloViz, "HoloViews," <https://holoviews.org/>.
- [23] --, "HvPlot," <https://hvplot.holoviz.org/>.
- [24] "Vue.js," 2014. [Online]. Available: <https://vuejs.org/>
- [25] M. Grinberg, *Flask web development: developing web applications with python*.^(o) Reilly Media, Inc.", 2018.
- [26] G. Van Rossum, *The Python Library Reference, release 3.8.2*. Python Software Foundation, 2020.
- [27] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *Journal of machine learning research*, vol. 12, no. Oct, pp. 2825-2830, 2011.
- [28] Google, "Google Colab," <https://colab.research.google.com/>, 2018.
- [29] A. Shreve, "Ngrok," <https://ngrok.com/>, 2012.

- [30] J. Shroff, R. Walambe, S. K. Singh, and K. Kotecha, "Enhanced security against volumetric ddos attacks using adversarial machine learning," *Wireless Communications and Mobile Computing*, 2022. [Online]. Available: <https://api.semanticscholar.org/CorpusID:247408815>
- [31] N. Bindra and M. Sood, "Detecting ddos attacks using machine learning techniques and contemporary intrusion detection dataset," *Automatic Control and Computer Sciences*, vol. 53, pp. 419 - 428, 2019. [Online]. Available: <https://api.semanticscholar.org/CorpusID:208086076>
- [32] M. M. Saeed, H. N. R. Mohammed, O. A. H. Gazem, R. A. Saeed, H. M. A. Morei, A. E. T. Eidah, A. S. A. Gaid, A. S. A. S. Al-Uosfi, and M. G. Q. Al-Madhagi, "Machine learning techniques for detecting ddos attacks," *2023 3rd International Conference on Emerging Smart Technologies and Applications (eSmarTA)*, pp. 1-6, 2023. [Online]. Available: <https://api.semanticscholar.org/CorpusID:264881735>
- [33] M. N. Faiz, O. Somantri, A. R. Supriyono, and A. W. Muhammad, "Impact of feature selection methods on machine learning-based for detecting ddos attacks : Literature review," *JOURNAL OF INFORMATICS AND TELECOMMUNICATION ENGINEERING*, 2022. [Online]. Available: <https://api.semanticscholar.org/CorpusID:246802887>
- [34] I. Sharafaldin, A. H. Lashkari, S. Hakak, and A. A. Ghorbani, "Developing realistic distributed denial of service (ddos) attack dataset and taxonomy," in *2019 International Carnahan Conference on Security Technology (ICCST)*. IEEE, 2019, pp. 1-8.
- [35] A. H. Lashkari, Y. Zang, G. Owahuo, M. Mamun, and G. Gil, "Cicflowmeter," *GitHub*. [vid. 2021-08-10]. Dostupn'e z: <https://github.com/ahlashkari/CICFlowMeter/blob/master/ReadMe.txt>, 2017.
- [36] L. Van der Maaten and G. Hinton, "Visualizing data using t-sne." *Journal of machine learning research*, vol. 9, no. 11, 2008.
- [37] J. H. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of statistics*, pp. 1189-1232, 2001.
- [38] H. Le, "CICFlowMeter," <https://gitlab.com/hieulw/cicflowmeter>, .2021
- [39] D. Ramos Archilla, "PyFlowmeter," <https://pypi.org/project/pyflowmeter/>, .2023

Annexes

Appendice A

6 source code of the project

A.1. preprocessing_dataset.py

```
import pandas as pd
import numpy as np

Import
from pathlib import Path

WORKING_DIR= Path.cwd()
DATA_UNPROCESSED= WORKING_DIR / "03-11"
DATA_PROCESSED= WORKING_DIR / "processed_files".

def transform_column_types(df):
    # Convert any float columns to float32 format
    df= df.apply(lambda x: x.astype("float32") if np.issubdtype(x.dtype, np.floating) else x)

    # Convert any int columns to int32 format
    df= df.apply(lambda x: x.astype("int32") if np.issubdtype(x.dtype, np.integer) else x)

    return df

def prerprocess_dataframe(df):
    start_mem_usg= df.memory_usage().sum() / 1024**2
    print("Memory usage of properties dataframe is :",start_mem_usg," MB") df.dropna(inplace=
    True)
    df.drop_duplicates(inplace= True)
    df= transform_column_types(df)
    final_mem_usg= df.memory_usage().sum() / 1024**2
    print("Memory usage of properties dataframe is :",final_mem_usg," MB")
    return df

def load_file(file_path):
    df= pd.read_csv(file_path,
                    usecols= lambda column: column not in columns_to_exclude)
    return prerprocess_dataframe(df)

columns_to_exclude= ['Unnamed: 0', 'Flow ID', ' Source IP', ' Source Port',
                    ' Destination IP', ' Destination Port', ' Timestamp', '
                    SimillarHTTP']

for file in os.listdir(DATA_UNPROCESSED):
    if file != 'MSSQL.csv' and file.endswith('.csv'):
        df= load_file(DATA_UNPROCESSED / file)
        basename= file.split('.')[0]
        df.to_parquet(DATA_PROCESSED / f"{basename}.parquet")
        print(file, '\n', df[' Label'].value_counts())
```

A.2. neuralnet.py

```

from sklearn.metrics import classification_report
from sklearn.model_selection import StratifiedKFold
from sklearn.utils.class_weight import compute_class_weight

import tensorflow as tf
from tensorflow.keras import layers, Model
from tensorflow.keras.optimisers import AdamW
from tensorflow.keras.losses import CategoricalCrossentropy
from tensorflow.keras import metrics

import numpy as np

N_FEATURES= X_train.shape[1]
N_CLASSES= len(label_encoder.classes_)

def create_model(n_features, n_classes):
    input= layers.Input(shape= (n_features, ))
    x= layers.Dense(64, activation= 'relu')(input)
    x= layers.Dense(64, activation= 'relu')(x)
    out= layers.Dense(n_classes, activation= 'softmax')(x)

    model= Model(inputs= input, outputs= out)

    learning_rate = 5e-4
    training_metrics= [
        metrics.Precision(name= 'Precision'),
        metrics.Recall(name= 'Recall'),
        metrics.F1Score(name= 'F1Score')
    ]

    model.compile( optimizer=
        AdamW(learning_rate), loss=
        CategoricalCrossentropy(), metrics=
        training_metrics
    )
    return model

bs= 32
epochs= 15

skf= StratifiedKFold(n_splits=5)
fold_idx = 0
for train_index, test_index in skf.split(X_train, y_train):
    print(f "Fold {fold_idx + 1}:")
    X_train_fold, X_test_fold= X_train[train_index, :], X_train[test_index, :]
    y_train_fold, y_test_fold= y_train[train_index], y_train[test_index]

    class_weights= compute_class_weight('balanced', classes= np.unique(y_train), y= y_train)
    class_weights= dict(enumerate(class_weights))
    model= create_model(N_FEATURES, N_CLASSES)
    history= model.fit(
        X_train_fold,
        tf.keras.utils.to_categorical(y_train_fold),
        validation_data= (X_test_fold, tf.keras.utils.to_categorical(y_test_fold)), batch_size
        bs,=
        class_weight= class_weights, epochs=
        epochs,
        verbose=0
    )

```

```

model.save(f "model_{fold_idx}.keras")
plt.figure()
plt.plot(history.history['loss'], label= 'Training Loss')
plt.plot(history.history['val_loss'], label= 'Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title(f'Fold {fold_idx+ 1} Loss')
plt.legend()
plt.show()

y_pred= model.predict(X_test_fold)
report= classification_report(y_test_fold,
                             np.argmax(y_pred, axis=1), target_names=
                             label_encoder.classes_)

print(report)
fold_idx+= 1

```

A.3. search_hyperparametros.py

```

import optuna
import xgboost as xgb
from sklearn.model_selection import cross_val_score

def objective(trial):
    params = {
        'objective': 'multi:softmax', # Multiclass classification
        'eval_metric': 'mlogloss',
        'device': 'gpu',
        'lambda': trial.suggest_float('lambda', 1e-8, 1.0, log= True),
        'alpha': trial.suggest_float('alpha', 1e-8, 1.0, log= True),
        'max_depth': trial.suggest_int('max_depth', 3, 9),
        'eta': trial.suggest_float('eta', 1e-8, 1.0, log= True),
        'gamma': trial.suggest_float('gamma', 1e-8, 1.0, log= True),
        'colsample_bytree': trial.suggest_float('colsample_bytree', 0.1, 1.0),
        'min_child_weight': trial.suggest_int('min_child_weight', 1, 10),
        'n_estimators': trial.suggest_int('n_estimators', 100, 1300),
        'num_class': len(set(Y)), # Number of classes
    }

    model= xgb.XGBClassifier(**params)
    model.fit(X_train, y_train)

    scores= cross_val_score(model, X_val, y_val, cv=5, scoring= 'accuracy')

    return scores.mean()

study= optuna.create_study(direction= 'maximize', study_name= 'Classifier')
study.optimize(objective, n_trials=100, show_progress_bar= True)

best_params= study.best_params
best_accuracy= study.best_value
print(f "Best Hyperparameters:
{best_params}") print(f "Best Accuracy:
{best_accuracy}")

```

A.4. pcap_ to csv pyflowmeter.py

```
from pyflowmeter.sniffer import create_sniffer

sniffer= create_sniffer(
    input_file= 'path_to_the_file.pcap',
    to_csv= True, output_file=
    './flows_test.csv',
)

sniffer.start()
try:
    sniffer.join()
except KeyboardInterrupt:
    print('Stopping the sniffer')
    sniffer.stop()
finally:
    sniffer.join()
```

A.5. send real traffic to server.py

```
from pyflowmeter.sniffer import create_sniffer

sniffer= create_sniffer(
    server_endpoint= 'http://127.0.0.1:5000/send_traffic', verbose=
    True,
    sending_interval=5
)

sniffer.start()
try:
    sniffer.join()
except KeyboardInterrupt:
    print('Stopping the sniffer')
    sniffer.stop()
finally:
    sniffer.join()
```

A.6. simulate traffic pyflowmeter.py

```
from pyflowmeter.sniffer import create_sniffer

sniffer= create_sniffer(
    input_file= 'path_to_the_file.pcap', server_endpoint=
    'http://127.0.0.1:5000/send_traffic',
)

sniffer.start()
try:
    sniffer.join()
except KeyboardInterrupt:
    print('Stopping the sniffer')
    sniffer.stop()
finally:
    sniffer.join()
```


A.7. endpoint start sniffer.py

```

@app.route('/start_sniffer', methods= ['POST'])
def start_sniffer():
    if request.is_json:
        data= request.get_json()
        test_type= data['file']
        test_file= TYPES_DICT[test_type]
        reload_sniffer(test_file)
        return jsonify({"message": "Data received successfully"}), 200
    else:
        return jsonify({"error": "Invalid JSON data in the request"}), 400

def reload_sniffer(test_file):
    global traffic_sniffer
    global sniffer_created
    global predicted_data
    if sniffer_created:
        try:
            traffic_sniffer.stop()
            traffic_sniffer.join()
        except:
            pass
    else:
        sniffer_created= True

    predicted_data= []
    if test_file== 'Real time traffic':
        traffic_sniffer= sniffer.create_sniffer(
            input_interface= 'eth0', server_endpoint=
                'http://127.0.0.1:5000/send_traffic',
        )
    else:
        traffic_sniffer= sniffer.create_sniffer( input_file=
            test_file, server_endpoint=
                'http://127.0.0.1:5000/send_traffic',
        )
    traffic_sniffer.start()

```

A.8. send_traffic and get_data.py

```
@app.route("/send_traffic", methods= ["POST"])
def post_data():
    if request.is_json:
        data= request.get_json()
        confidences, predcted_classes= model.predict(data["flows"])
        for (flow, confidence, predcted_class) in zip(
            data["flows"], confidences, predcted_classes
        ):

            predicted_data.append(
                {
                    "type": predcted_class,
                    "src_ip": f'{flow["src_ip"]}:{flow["src_port"]}',
                    "dst_ip": f'{flow["dst_ip"]}:{flow["dst_port"]}',
                    "confidence": f'{confidence:.2%}',
                    "timestamp": flow["timestamp"],
                }
            )

        print(model.predict(data["flows"]))
        return jsonify({"message": "Data received successfully"}), 200
    else:
        return jsonify({"error": "Invalid JSON data in the request"}), 400

@app.route("/get_data", methods= ["GET"])
def get_data():
    return jsonify(predicted_data)
```

A.9. router.js

```
import { createRouter, createWebHistory } from 'vue-router'
import Dashboard from '../components/Dashboard.vue'.
import TrafficAnalysis from '../components/TrafficAnalysis.vue'.

const router= createRouter({
  history: createWebHistory(import.meta.env.BASE_URL),
  routes: [
    {
      path: '/dashboard',
      name: 'dashboard',
      component: Dashboard
    },
    {
      path: '/traffic-analysis',
      name: 'traffic-analysis',
      component: TrafficAnalysis
    }
  ]
})

export default router
```

A.10. firewall model.py

```

from joblib import load
import numpy as np
import pickle
import tensorflow as tf

class FirewallModel:

    TF_MODEL_FILE_PATH= ".//ML_models/model.keras"
    XGB_MODEL_FILE_PATH= ".//ML_models/xgboost_model.pkl"
    SCALER_PATH= ".//ML_models/std_scaler.bin"
    LABEL_ENCODER_PATH= ".//ML_models/label_encoder.bin"
    COLUMNS_ORDER [=
        "protocol",
        "flow_duration",
        "tot_fwd_pkts",
        "tot_bwd_pkts",
        ... # Omitted for memory
        "idle_min",
    ]

    def init (self, use_xgboost= True):
        self.use_xgboost= use_xgboost
        self.model= self.load_model()
        self.scaler= self.load_scaler()
        self.label_encoder= self.load_label_encoder()

    def load_model(self):
        if self.use_xgboost:
            with open(FirewallModel.XGB_MODEL_FILE_PATH, "rb") as model_file:
                return pickle.load(model_file)
        else:
            return tf.keras.models.load_model(FirewallModel.TF_MODEL_FILE_PATH)

    def load_scaler(self):
        return load(FirewallModel.SCALER_PATH)

    def load_label_encoder(self):
        return load(FirewallModel.LABEL_ENCODER_PATH)

    # returns the class and probability (confidence in prediction)
    def predict(self, data):
        prepared_data= self.prepare_data(data)
        print(prepared_data.shape)
        if self.use_xgboost:
            preds= self.model.predict_proba(prepared_data)
        else:
            preds= self.model.predict(prepared_data)
        return preds[
            np.arange(preds.shape[0]), predicted_classes
        ], self.label_encoder.inverse_transform(predicted_classes)

    def prepare_data(self, data):
        final_data= []
        for flow in data:
            features_list= [flow[feature] for feature in FirewallModel.COLUMNS_ORDER]
            final_data.append(features_list)
        return self.scaler.transform(np.array(final_data))

```